SOLIDIFIED

Audit Report for Galleon DEX - April 27, 2021

Summary

Audit Report prepared by Solidified covering the Galleon DEX smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code in several rounds. The debrief of round 1 took place on 29 March 2021.
The debrief of round 2 took place on 27 April 2021, and the results are presented here.

Audited Files

The source code has been supplied in the form of a GitHub repository:

https://github.com/shipyard-software/galleon-dex

Commit number: `0dff47ee8c15a6375e93365efa4dd3f4617201cb`

The scope of the audit was limited to the following files:

```
contracts
├── BlacklistAndTimeFilter.sol
├── GalleonDeposit.sol
├── GalleonEscapeContract.sol
├── GalleonExchangeInterface.sol
├── GalleonPool.sol
└── libraries
    ├── AggregatorInterface.sol
    ├── ApprovalInterface.sol
    ├── Sqrt.sol
    └── UniERC20.sol
```

Intended Behavior

The smart contracts implement an automated market maker aimed at providing a decentralized exchange.

SOLIDIFIED

Audit Report for Galleon DEX - April 27, 2021

Code Complexity and Test Coverage

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does equate to a higher risk. Certain bugs are more easily detected in unit testing than a security audit and vice versa. It is, therefore, more likely that undetected issues remain if the test coverage is low or non-existent.

Criteria	Status	Comment
Code complexity	Medium-High	-
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	Medium	-

Test coverage report:

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/	92.31	83.33	100	92.86	
AggregatorInterface.sol	100	100	100	100	
ApprovalInterface.sol	100	100	100	100	
Sqrt.sol	100	100	100	100	
UniERC20.sol	92.31	83.33	100	92.31	39
contracts/mocks/	75	100	71.43	75	
MockOracle.sol	66.67	100	50	66.67	20
MockToken.sol	75	100	75	75	29
SqrtMock.sol	100	100	100	100	
All files	82.11	51.28	65	80.6	

SOLIDIFIED

Audit Report for Galleon DEX - April 27, 2021

Issues Found

Solidified found that the Galleon Dex contracts contain no critical issues, 2 major issues, 3 minor issues, in addition to 7 informational notes.

We recommend all issues are amended, while the notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	Iterations over variable-sized data structure may cause critical functions to fail if too many tokens registered	Major	Acknowledged
2	GalleonExchangeInterface.sol: some functions can be blocked by a denial of service attack or malfunctioning token	Major	Acknowledged
3	GalleonPool.sol: Non-enforcement of ETH as last element in the linked list may break escape protection	Minor	Resolved
4	GalleonPool.sol: Missing zero-checks	Minor	Pending
5	UniERC20.sol: Functions uniTransfer()/uniTransferFromSender() can potentially fail when transferring ETH to a smart contract	Minor	Pending
6	Consider using additional events	Note	-
7	Use constants instead of magic numbers	Note	-
8	Use modifier instead of copying require constraints	Note	-
9	GalleonPool.sol: _escapeContract doesn't need owner restriction	Note	-
10	GalleonExchangeInterface.sol: potential gas optimization	Note	-

11	GalleonExchangeInterface.sol: add check for tradebility in _sync()	Note	-
12	GalleonDeposit.sol: Unnecessary gas expenditure by declaring myDeposit as memory instead of storage	Note	-

Critical Issues

No critical issues have been found.

Major Issues

1. Iterations over variable-sized data structure may cause critical functions to fail if too many tokens registered

The `GalleonPool` contract stores `asset` in a linked list. There are several functions that iterate over these data structures:

`GalleonPool.sol`:
- `removeToken()`
- `syncAll()`

`GalleonExchangeInterface.sol`:
- `withdraw()`
- `invariant()`

If this data structure grows too large, due to many tokens being registered with the pool, these iterations may hit the block gas limit, leading to the transactions always reverting.

Recommendation
Consider using a data model that does not require looping over variable-sized data structures. It seems the linked list implementation is not really required to keep track of all tokens and removing it would also provide significant gas savings.

Round Two Update
The team acknowledges this property. In this particular case, the iteration is considered acceptable, since the smart contract will only be used for a limited number of assets. The addition and removal of assets are operator-controlled.
Solidified recommends coding a maximum limit into the smart contract, in order to avoid exceeding the safe limit accidentally.

2. GalleonExchangeInterface.sol: some functions can be blocked by a denial of service attack or malfunctioning token

Throughout the code, external calls are performed to registered tokens, for instance in `withdraw()` and `syncAndTransfer()`. If an external token misbehaves by reverting, the whole transaction will fail. This can be exploited by malicious tokens that revert to perform a denial of service attack.

Recommendation
Consider token withdrawals to be performed individually and/or use `try` and `catch` clauses to prevent transactions from failing completely.

Round Two Update
The team acknowledges this property. All tokens will be vetted by the operators before adding them.

Minor Issues

3. GalleonPool.sol: Non-enforcement of ETH as the last element in the linked list may break escape protection

The `escape()` function protection relies on assuming the ETH token entry will be inserted first. However, there is nothing to enforce in the codebase that really is placed at this point in the data structure by the admin team. If this assumption is violated accidentally or on purpose, the protection mechanism in the `escape()` function will not work.

Recommendation
Consider including checks to ensure that ETH is inserted as the first element.

Round Two Update
This issue has been resolved through refactoring.

4. GalleonPool.sol: Missing zero-checks

The functions `modifyDepositContract()`, `modifyApprovalContract()` and `modifyExchangeInterfaceContract()` do not check for `address(0)`. This may cause protocol malfunctioning if these functions are called with zero arguments.

Recommendation

Consider adding zero-checks.

5. UniERC20.sol: Functions uniTransfer()/uniTransferFromSender() can potentially fail when transferring ETH to a smart contract

Function `uniTransfer()` & `uniTransferFromSender()` call `transfer()` when sending ETH to `to`/`sendTo`, which only forwards 2300 gas. In cases where `sendTo` address is a smart contract whose fallback function consumes more than 2300 gas, the call will always fail. This will have the side effect of potentially preventing smart contracts (e.g. DAOs) from receiving transfers.

For a more in-depth discussion of issues with `transfer()` and smart contracts, please refer to `https://diligence.consensys.net/blog/2019/09/stop-using-soliditys-transfer-now/`

Recommendation
Replace instances of `transfer()` with `call()`.

Informational Notes

6. Consider using additional events

It is good practice to emit an event when updating key protocol parameters or adding an asset. The current implementation does not use many event types. For instance, no events are emitted when the fees are changed or a new asset is added.

Recommendation
Consider adding event types

7. Use constants instead of magic numbers

Much of the code has hard-coded numbers instead of declared constants. For example, the multiplier value, the token decimals, and the number of seconds in `ActivateRemoval`.

Recommendation
Consider using declared top-level constants to replace the magic numbers.

8. Use modifier instead of copying require constraints

In many places, the code repeats certain access constraints as pre-conditions for functions. It would be cleaner to use modifiers for this instead of copying the constraint.

For instance in the following functions:

- `recordDeposit`
- `recordUnlockedDeposit`
- `syncAll`
- `sync`
- `transfer`
- `syncAndTransfer`
- `swapAndBurn`

Recommendation
Consider using modifiers.

9. GalleonPool.sol: _escapeContract() doesn't need owner restriction

This function is just a view function so it does not require a calling restriction, since it changes no state.

Recommendation
Consider removing the restriction.

10. GalleonExchangeInterface.sol: potential gas optimization

The function `invariant()` could benefit from CALL reduction
Creating a `getTokenDetails(address token)` function that returns a tuple of `(oracle, marketShare, lastBalance)` for a token would reduce the CALL and stack overhead costs incurred each time the contract calls out to GalleonPool.

Recommendation
Consider adding this function to optimize gas.

11. GalleonExchangeInterface.sol: add check for tradebility in _sync()

The function `_sync()` relies on a nested check in function `balancesAndMultipliers()` to validate input parameters:

```
require(isTradable(inputToken) && isTradable(outputToken), "Galleon: Untradable asset(s)");
```

This does not favor the readability and maintainability of the code.

Recommendation
Consider placing this check in `_sync()`

12. GalleonDeposit.sol: Unnecessary gas expenditure by declaring myDeposit as memory instead of storage

Function `unlockVestedDeposit()` assigns `deposits[msg.sender]` to a `memory` variable, which results in the value being copied from the contract's storage to memory, wasting unnecessary gas.
Function `deposit()` also has the same issue with `curDeposit`.

Recommendation
Declare `myDeposit` as a `storage` variable so that no unnecessary copying takes place.

Audit Report for Galleon DEX - April 27, 2021

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Galleon Dex or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.