SOLIDIFIED

Audit Report for Galleon DEX - March 29, 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 below. The final debrief took place on 29 March 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: `ffe109e6406fc90e37c99d627ec4abd266840a6a`

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.

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-Low	-

Test coverage report:

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	59.8	31.25	51.67	58.62	
BlacklistAndTimeFilter.sol	40	100	30	36.36	... 43,47,51,55
GalleonDeposit.sol	59.09	30	75	56.52	... 93,94,95,99
GalleonEscapeContract.sol	33.33	0	50	33.33	19,20
GalleonExchangeInterface.sol	60	16.67	64.29	60	... 210,213,216
GalleonPool.sol	62.92	40.91	50	61.54	... 312,316,319
contracts/libraries/	76	55.56	80	75	
AggregatorInterface.sol	100	100	100	100	
ApprovalInterface.sol	100	100	100	100	
Sqrt.sol	83.33	66.67	100	81.82	13,22
UniERC20.sol	69.23	50	75	69.23	20,21,23,43
contracts/mocks/	66.67	100	60	66.67	
MockOracle.sol	66.67	100	50	66.67	20
MockToken.sol	66.67	100	66.67	66.67	29
All files	61.74	35.71	54.29	60.52	

SOLIDIFIED

Audit Report for Galleon DEX - March 29, 2021

Issues Found

Solidified found that the Galleon Dex contracts contain no critical issue, 2 major issus, 2 minor issues, in addition to 5 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	Pending
2	GalleonExchangeInterface.sol: some functions can be blocked by a denial of service attack or malfunctioning token	Major	Pending
3	GalleonPool.sol: Non-enforcement of ETH as last element in linked list may break escape protection	Minor	Pending
4	GalleonPool.sol: Missing zero-checks	Minor	Pending
5	Consider using additional events	Note	-
6	Use constants instead of magic numbers	Note	-
7	Use modifier instead of copying require constraints	Note	-
8	GalleonPool.sol: _escapeContract doesn't need owner restriction	Note	-
9	GalleonPool.sol: Recording lastETHBalance is unnecessarily expensive	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()`

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 seem the linke list implementation is not really required to keep of track of all tokens and removing it would also provide significant gas savings.

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 tokens 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.

Minor Issues

3. GalleonPool.sol: Non-enforcement of ETH as last element in 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.

4. GalleonPool.sol: Missing zero-checks

The functions `modifyDepositContract()` 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.

Informational Notes

5. 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 emmit when the fees are changed or a new asset is added.

Recommendation

Consider adding event types

6. Use constants instead of magic numbers

Much of the code has hardcoded 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.

7. Use modifier instead of copying require constraints

In many places the code repeats certain access constraints as pre-conditions for funcitons. 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.

8. 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.

9. GalleonPool.sol: Recording lastETHBalance is unnecessarily expensive

It seems that the `lastETHBalance` variable is not needed. It appears to be used as a convenience variable. However, calling into another contract to read the balance is cheaper than writing to storage using 20k gas each time.

Recommendation

Consider removing the variable and related bookkeeping.

Audit Report for Galleon DEX - March 29, 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.