SOLIDIFIED

Audit Report for Vega Protocol - October 11, 2021

Summary

Audit Report prepared by Solidified covering the Vega Protocol 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 October 11, 2021, and the results are presented here.

Audited Files

The source code has been supplied in 3 private code repositories:

https://github.com/vegaprotocol/Claim_Codes/tree/feat/v2
Commit number: `b4724ecda8fdd636251966b42752f51205bc63b8`

https://github.com/vegaprotocol/MultisigControl
Commit number: `bba9dad49623bb297dc0b35d1a8491bc6d691841`

https://github.com/vegaprotocol/Staking_Bridge
Commit number: `c62309dfd40a0eb211a0102c63e191a47b9e46ec`

UPDATE: Fixes were received on October 14, 2021 and the report was updated accordingly.

Intended Behavior

Vega Protocol is a decentralized network for trading derivatives.

Findings

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 not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

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

Issues Found

Solidified found that the Vega Protocol contracts contain no critical issues, 2 major issues, 2 minor issues, 6 informational notes and 1 warning.

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

Issue #	Description	Severity	Status
1	ERC20_Asset_Pool.sol: Function withdraw() assumes all tokens return a value on transfer	Major	Resolved
2	ETH_Asset_Pool.sol: Contract will always fail to receive ETH	Major	Resolved
3	ETH_Asset_Pool.sol: Function withdraw() can potentially fail when transferring ETH to a smart contract	Minor	Resolved
4	ETH_Asset_Pool.sol / ERC20_Asset_Pool.sol: Contracts could potentially be assigned an invalid multisig_control_address	Minor	Resolved
5	Vega_Staking_Bridge.sol: Function transfer_stake() does not validate new_address	Note	Acknowledged
6	MultisigControl Contracts: gas usage could be optimized	Note	Resolved
7	ERC20_Bridge_Logic.sol: Contract unnecessarily uses the SafeMath library	Note	Resolved
8	MultisigControl.sol: Events do not conform to the IMultisigControl interface	Note	Resolved
9	ETH_Asset_Pool.sol: Contract's ETH balance is not checked before initiating a transfer	Note	Resolved
10	Miscellaneous Notes	Note	Acknowledged
11	Claim_Codes.sol: Allowed countries are easily gameable	Warning	Acknowledged

Critical Issues

No critical issues have been found.

Major Issues

1. ERC20_Asset_Pool.sol: Function withdraw() assumes all tokens return a value on transfer

The function `withdraw()` assumes that all tokens' `transfer()` function returns a value, which is not true for a lot of common tokens that don't fully conform to the `ERC20` standard. If the deposited token's `transfer()` function does not return a value, `withdraw()` will always fail, and the funds will remain forever locked in the contract.

Recommendation

Consider using Open Zeppelin's `SafeERC20.safeTransfer()` instead of `transfer()`.

Note 1

The same issue applies to `transferFrom()` in `ERC20_Bridge_Logic.deposit_asset()`, albeit to a lesser degree of severity.

Note 2

For a list of common tokens that don't return a value on `transfer()`, please refer to `https://medium.com/coinmonks/missing-return-value-bug-at-least-130-tokens-affected-d67bf08521ca`

Status

Resolved

2. ETH_Asset_Pool.sol: Contract will always fail to receive ETH

The `ETH_Asset_Pool` contract is supposed to receive ETH whenever the `ETH_Bridge_Logic.deposit_asset()` is called. However, `ETH_Asset_Pool` doesn't have a `receive()` function, and thus won't be able to receive any ETH due to the transaction always getting reverted.

Recommendation
Implement a `receive()` function in order to allow `ETH_Asset_Pool` to receive ETH.

Status
Resolved

Minor Issues

3. ETH_Asset_Pool.sol: Function withdraw() can potentially fail when transferring ETH to a smart contract

Function `withdraw()` calls `transfer()` when sending ETH to `target`, which only forwards 2300 gas. In cases where the `target` 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()`.

Status

Resolved

4. ETH_Asset_Pool.sol / ERC20_Asset_Pool.sol: Contracts could potentially be assigned an invalid multisig_control_address

---

In the case where `ETH_Asset_Pool` and `ERC20_Asset_Pool` contracts are mistakenly assigned an invalid `multisig_control_address`, the contracts will be permanently locked out of setting a new `multisig_control_address` or `ETH_bridge_address`.

Recommendation

Require that `multisig_control_address != address(0)`.

Status

Resolved

Informational Notes

---

5. Vega_Staking_Bridge.sol: Function transfer_stake() does not validate new_address

---

Lack of validation for `new_address` can result in users accidentally burning their staked tokens.

Recommendation

Consider requiring that `new_address != address(0)`.

Status

Acknowledged. Team's response: "*No fix for this has been implemented at this time. The contract is live in production*".

6. MultisigControl Contracts: gas usage could be optimized

The `MultisigControl` contracts all declare their respective functions' `signatures` array as `memory`, which consumes extra gas as the compiler copies its value from `calldata`.

Recommendation

Consider either declaring `signatures` as `calldata` instead of `memory`, or declaring all functions as `external` instead of `public`.

Status

Resolved

7. ERC20_Bridge_Logic.sol: Contract unnecessarily uses the SafeMath library

---

Since Solidity versions 8.0 and later automatically revert on arithmetic underflow and overflow, there is no need to use the `SafeMath` library in `ERC20_Bridge_Logic`.

Recommendation

Consider removing the `SafeMath` library to save on gas.

Status

Resolved

8. MultisigControl.sol: Events do not conform to the IMultisigControl interface

The `MultisigControl` contract declares the `SignerAdded`, `SignerRemoved` and `ThresholdSet` events with only one parameter, while `IMultisigControl` declares them with two parameters.

Recommendation
Consider having only one version of the aforementioned events.

Status
Resolved

9. ETH_Asset_Pool.sol: Contract's ETH balance is not checked before initiating a transfer

If the transferred ETH is more than the current contract's balance, the transaction will revert without giving an adequate error message.

Recommendation
Consider checking the contract's balance before transferring and providing an error message if the amount exceeds the contract's balance.

Status
Resolved

10. Miscellaneous Notes

- Vega_Staking_Bridge.sol: Functions `remove_stake()` & `transfer_stake()` lack informational error messages when user input exceeds their stake.
- Claim_Codes.sol, Vega_Staking_Bridge.sol: several `require()` statements are missing their respective error messages.
- Claim_Codes.sol: Function `permit_issuer()` is missing zero address validation.

Status

Acknowledged. Team's response: "*No fix for this has been implemented at this time. The contract is live in production*".

Warnings

11. Claim_Codes.sol: Allowed countries are easily gameable

Users are required to enter their country code when claiming a code in functions `claim_targeted()` and `claim_untargeted()`, however this is impossible to enforce.

Status

Acknowledged. Team's response: "*It is understood that the countries are gameable, this is not fixable but does show ill intent of the gamer*".

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Vega Protocol 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.