Audit report for SportX

DECENTER

1. Summary

SportX has engaged Decenter in the period starting on January 20th and ending on February 2nd 2020 to assess and audit their platform's Solidity smart contracts. This document describes the issues discovered during the audit. Decenter's assessment is focused primarily on code review of the smart contracts, with an emphasis on security, gas usage optimization and overall code quality.

2. Audit

2.1 Authenticity

The audited contracts can be found in the Gitlab repository: https://gitlab.com/nextgenbt/betx/betx-contracts/tree/development; the version used has commit hash *799584d8c49f4d1a0c1b3818b1defe549c224a09*.

2.2 Scope

This audit covered only the *.sol files mentioned in the previous section.

2.3 Legend

🔴 High priority issues

🟠 Medium priority issues

🟡 Low priority issues and optimisations

🟢 Notes and recommendations

3. Issues Found

1. Cancelled orders could still be filled

File name: FillOrder.sol;

The CancelOrder.sol smart contract includes on-chain logic that enables orders to be cancelled. Only orders where the executor is not set can be cancelled and they can be cancelled only by the maker of that order. Once an order is cancelled other users should not be able to fill that order any more. However, there is no on-chain check in the fillOrder logic if the order has been cancelled, allowing orders that have been cancelled to be filled.

We recommend adding an additional check in the fillOrder logic so that orders that are cancelled cannot be filled.

2. No need to define zeroAddress variable

File name: LibOrder.sol (lines 52,77);

In two places in the LibOrder library an empty (0x0) address is defined in a variable. Solidity already has this pre-defined as `address(0)` which resolves to an empty Ethereum address, so there is no need to explicitly define it.

3. A constant used as local variable

File name: LibOrder.sol (line 34);

The `oddsPrecision` variable is declared as a local variable even though it is in fact a constant. To improve code readability and save on gas usage, we recommend declaring it as a constant variable outside the scope of the function.

4. Old defaultOperator will still have allowance in WETH token

File name: WETH.sol;
(line 260) _deposit() function;

In the modified version of the WETH token, each time a user makes an Ether deposit to the contract, an allowance is set to an address referred to as the defaultOperator. This is an intentional feature of the modified WETH contract, which the user should be aware of. The default operator can however be changed by the admin (defaultOperatorController) at any point in time. If the address of the defaultOperator changes, the allowance the old defaultOperator had will still remain in place, so the admin of the contracts needs to be aware of this. If, for instance, the defaultOperator address is compromised, the admin would need to switch to a new default operator, as well as zero out all the given allowances that the old default operator had.

5. Transfer of non-standard base tokens will fail

File name: Escrow.sol;
(lines 127-130) settleBet() function;

In the settleBet() function, which handles the payouts of the bets after they are resolved, the payouts are made in a baseToken which is specified by the order. The baseToken does a .transfer() call to move the funds and expects that the method returns a boolean true/false value of whether the transfer succeeded. Some tokens, such as the OMG (OmiseGo) token, don't return a boolean value on transfer(), so the creator of the orders should be careful that the baseTokens that are being used are fully compliant to the ERC20 token standard.

6. Incorrect notice description

File name: OutcomeReporter.sol (lines 48-51);

The @notice description for reportOutcome() states that the reported market can be challenged and voted on, which is not true based on the code implementation.

7. Use of magic numbers in code

File name: LibOrder.sol (lines 54,83);

In two places a magic number `10**20` is used. It is good coding practice not to hardcode constants as numbers, but to define them as named constants instead.

8. Making variables private and then explicitly implementing a getter function

File name: SystemParameters.sol, OutcomeReporter.sol, AffiliateRegistry.sol, Fills.sol, FeeSchedule.sol;

In multiple contracts (listed above) variables are made private only to later have explicitly defined getter functions for them. In Solidity public variables have a getter generated by default, so to make variables private only to later create getters is redundant.

It should be noted that making variables private in Solidity does not mean that the content they hold cannot be read by other entities, due to the nature of the blockchain everything remains public and visible. Making the variables private only means that other on-chain contracts cannot access the data.

9. Use of experimental features

The contracts use an experimental `ABIEncoderV2` feature to properly handle structs as input/output data in functions. While this does improve useability of the code, using experimental features in production is not the best security practice.

4. Audit

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of business model, or any other statements about fitness of the contracts to purpose or their bugfree status. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

5. Closing Summary

It is the conclusion of this review that the codebase of the platform is well organized and tested. The code is fully covered by the documentation which is well written and helps with general readability and understanding of the code. Besides the medium priority issue, all the other issues are low priority, and they are mainly there to point out to the developer some potential edge cases and problems that might arise in the future. The notices are there to further improve code readability and support good coding practices. We strongly advise that the frontend code is written in such a way that it also performs checks in order to stop the user from interacting with the code in any unexpected manner. The code is optimised and no significant gas savings can be done without changing the code too much. Aside from the remarks in this audit, the security of the contracts is sufficient given the assumed requirements.