DexSport Security Analysis

by Pessimistic

This report is public

28 January, 2022

Abstract

In this report, we consider the security of smarts contracts of DexSport project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, security audit is not an investment advice.

Summary

In this report, we considered the security of DexSport smart contracts. We performed our audit according to the procedure described below.

The audit showed several medium severity issues, including Overpowered owner, Locked ether, and Unconventional upgrade logic. Also, many issues of low severity were found.

The project has no public documentation.

After the initial audit, the code base was updated. Multiple issues were fixed, and several new were discovered.

Later the code base was updated again. Most of known issues were fixed, however, more issues were discovered in the updated and new code, including Underflow and two Bugs.

Third update fixed some of previously discovered issues. However, one of the bugs wasn't handled properly, which resulted in a new Bug.

The code base was updated once more. The last Bug was fixed, and some minor improvements were introduced.

A minor update slightly improved code quality. A later update addressed the last of code issues.

General recommendations

We recommend adding public documentation to the project and providing NatSpec comments to the code base.

Project overview

Project description

For the audit, we were provided with **bsc.sol** file, SHA1 checksum `CC1960628AFF0F5FD72FEA54269BD31CAF28D30C`.

No project or test files were provided, therefore code coverage is not assessed. The project has a private documentation, and the code has no NatSpec comments.

The total LOC of audited sources is 206.

Code base update

After the initial audit, the code base was updated. For the recheck, we were provided with **dexsport_contracts-master.zip** archive, SHA1 checksum `905271BB12722D33FED6D04ACF0BE9BB20351547`.

It includes project files and tests, all seven of which pass. The resulting code coverage is 34.94%, however, main contract's logic is not covered by tests.

Note that **DexBets** contracts was renamed to **DexSportMain**.

Code base update #2

After the first recheck, the code base was updated again. We were provided with **dexsport_contracts-master-3.zip** archive this time, SHA1 checksum `61C0C68467812AC467B640819C1A4FBE3DA3111C`.

This update introduces `Dao` contract with multisig functionality. It protects the most critical functions of `DexSportMain` contract.

Besides the code, it includes project files and tests, all nine of which pass. The code coverage estimation is set up and is 100%. Additionally, the project has everything configured to run Slither.

Code base update #3

Later the code base was updated again. We were provided with **dexsport_contracts-master-5.zip** archive this time, SHA1 checksum `39C6CCBA09B369009E30F7C9A24FEBAB3705DB6D`.

This update fixes previously discovered issues and removes multiple unused contracts.

Code base update #4

The code base was updated again. We were provided with **dexsport_contracts-10.zip** archive this time, SHA1 checksum `1DACF85200CA90BDB75D2ABF1F33863D4A6D9D0F`.

This update fixes previously discovered issues.

Code base update #5

The code base was updated again. We were provided with **dexsport_contracts-11.zip** archive this time, SHA1 checksum `9AB7F1CCF27193AFD19A933D3D599DA4F8FA73C3`.

This update fixes previously discovered issues.

Code base update #6

The code base was updated again. We were provided with **dexsport_contracts-12.zip** archive this time, SHA1 checksum `526EF7E825DF866EFB1271761561984C10A237AB`.

This minor update removes ambiguity in the comments to **DexSportMain** contract.

Procedure

In our audit, we consider the following crucial features of the code:

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- Automated analysis
  - We scan project's code base with automated tools: Slither and SmartCheck.
  - We manually verify (reject or confirm) all the issues found by tools.
- Manual audit
  - We manually analyze code base for security vulnerabilities.
  - We assess overall project structure and quality.
- Report
  - We reflect all the gathered information in the report.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger smart contracts security. We highly recommend fixing them.

The audit showed no critical issues.

Medium severity issues

Medium issues can influence project operation in current implementation. We highly recommend addressing them.

Overpowered owner (fixed)

Most actions in the system require a transaction from the owner. The owner of **DexBets** contract has full control over users' bets and tokens.

In the current implementation, the system depends heavily on the owner of the contract. Thus, there are scenarios that can lead to undesirable consequences for the project and its users, e.g., if the owner's private keys become compromised.

We recommend designing contracts in a trustless manner or implementing proper key management, e.g., multisig.

Most critical functions are protected by musltisig-style contract.

Unconventional upgrade logic (fixed)

Once the **DexBets** contract becomes deprecated as a part of system upgrade, users lose an ability to retrieve their bets and rewards via `withdrawal()` function. Usually, upgrade logic prevents new users from joining the protocol, while allowing everyone to withdraw their assets.

The issue has been fixed and is not present in the latest version of the code.

Locked ether (fixed)

`withdrawal()` function of **DexBets** contract accepts ether. However, there is no way to withdraw it from the contract. Consider removing `payable` keyword.

The issue has been fixed and is not present in the latest version of the code.

Tests (fixed)

The logic of **DexSportMain** contract is not covered by tests.

New tests were added, code coverage is 100%.

Underflow (fixed)

Since the contracts were downgraded to Solidity 0.7.6, compiler doesn't insert over- and underflow checks for arithmetic operations. Underflow can happen in multiple functions of **DexSportMain** contract, it affects correctness of reserves feature.

The code was updated to Solidity 0.8.4, the issue is not present in the latest version of the code.

Bugs (fixed)

- The `isTransferAvailable` and `isOwnerChangeAvailable` functions of **Dao** contract ignore votes from the first voter. They have `0` index internally, however, the functions iterate over voters starting with `1` index.
- Reserve amount increase by the bet's `reserve` value for each `newbet` call to **DexSportMain** contract. However, bet's `amount` is subtracted from the reserves when the bet is updated by the platform via `toPayAdmin` call.
- `toPayAdmin` function of **DexSportMain** contract updates reserves amounts among other things. However, the code at lines 131-132 is bugged since it doesn't change reserves at all.
  The line 136 reduces the reserve by the bid's `toPay` value, which is 0 initially. So it doesn't make sense, unless it is the second time when the owner updates this bet, i.e. they are changing the bet from winner to loser status.

The issues have been fixed and are not present in the latest version of the code.

Low severity issues

Low severity issues can influence project operation in future versions of code. We recommend taking them into account.

Code quality (fixed)

- Mark functions as `external` where possible in the **DexBets** contract.

  The issue has been fixed and is not present in the latest version of the code.

- In **DexBets** contract, when comparing a variable against a boolean value, consider using `!X` syntax instead of `X == false` at lines 202 and 236.

  The issue has been fixed and is not present in the latest version of the code.

- Consider declaring `UpgradedContract` as `interface`.

  The code has been updated, which removed this issue.

- Mark functions as `external` where possible in the **DexBets** contract.

  The issue has been fixed and is not present in the latest version of the code.

- Storage variables are considered `internal` by default. Declare visibility of **DexBets**'s variables explicitly to improve code readability.

  The issue has been fixed and is not present in the latest version of the code.

- `withdrawal()` function of **DexSportMain** contract violates CEI pattern.

  The issue has been fixed and is not present in the latest version of the code.

- There are multiple view functions that will revert if the contract is paused. We don't recommend reverting calls to view functions, consider removing `whenNotPaused` modifier.

  The code has been updated, which removed this issue.

- Reserve underflow check at line 184 of **DexSportMain** contract is ineffective, since reserves are of `uint` type, and compiler automatically inserts overflow checks. Consider removing this line.

  The issue has been fixed and is not present in the latest version of the code.

- `Withdrawn` event is declared but never emitted in `DexSportMain`.

  The issue has been fixed and is not present in the latest version of the code.

- `Bet` struct is initialized line by line at lines 92-97 of **DexSportMain**'s `newbet` function. Besides, `version` field is left uninitialized. Consider initializing structs with a single assignment, i.e., `MyStruct ms = MyStruct(x, y);` or `MyStruct ms = MyStruct({a:x, b:y});`.

> *Comment from developers: We do it according to lower gas.*

Erc20 (fixed)

DexBets contract works with USDT, which does not fully comply with ERC20 standard, since `transfer()`, `transferFrom()`, and `approve()` functions of USDT contract always return `false`). **DexBets** does not comply with ERC20 standard as well, since its code does not check return values of calls to USDT. These issues cancel each other, therefore we only recommend changing `IERC20` interface so it correctly models USDT functions.

The issue has been fixed and is not present in the latest version of the code.

Gas consumption (fixed)

- Consider declaring `usdt` storage variable of the **DexBets** contract as `immutable`.

  The code has been updated, which removed this issue.

- `reserved` storage variable of **DexBets** contract is updated inside loops at lines 226 and 238. Consider updating it only once to reduce gas consumption.

  The issue has been fixed and is not present in the latest version of the code.

- Consider declaring `max_amount` in **DexBets** as a constant.

  The code has been updated, which removed this issue.

- Optimization is disabled (not enabled explicitly) in **hardhat.config.js** config file.

  The issue has been fixed and is not present in the latest version of the code.

Off-chain integration (fixed)

Consider declaring some event parameters as `indexed` to improve integration with off-chain code. E.g. `id` and `user` parameters in **DexBets** events.

The code has been updated, which removed this issue.

No public documentation

The project has no public documentation. The contracts are well covered with comments, however, they are not in NatSpec format.

Considering the complexity of the project, the documentation is important not only for the audit but also for development process. It should explicitly explain the purpose and behavior of the contracts, their interactions, and main design choices.

Code style (fixed)

The code does not follow Solidity Style Guide.

The issue has been fixed and is not present in the latest version of the code.

Notes

Upgradability (fixed)

DexSportV1 and **DexSportV2** contracts have incompatible storage layout. It is a major issue for most upgradability patterns.

The issue has been fixed and is not present in the latest version of the code.

USDT integration (fixed)

DexSportMain contract correctly works with ERC20 tokens. However, a comment in the beginning of the contract mentions USDT token, which is not ERC20 compatible, and the system will not work correctly with it.

The issue has been fixed and is not present in the latest version of the code.

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Vladimir Tarasov, Security Engineer
Boris Nikashin, Analyst
Alexander Seleznev, Founder

28 January, 2022