SmartDec

Orbs Smart Contracts Security Analysis

This report is public.

Published: February 27, 2020.

Abstract

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

The audit showed neither critical nor medium severity issues. However, a number of low severity issues were found.

All of the issues were fixed in the latest version of the code.

General recommendations

The contracts code is of high code quality. Therefore, we do not have any additional recommendations.

Checklist

Security

The audit showed no vulnerabilities.
Here by vulnerabilities we mean security issues that can be exploited by an external attacker. This does not include low severity issues, documentation mismatches, overpowered contract owner, and some types of bugs.

Compliance with the documentation

The audit showed no discrepancies between the code and the provided documentation.

Tests

The audit showed that the code was covered with tests sufficiently.

The text below is for technical use; it details the statements made in Summary and General recommendations.

Procedure

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

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices in efficient use of gas, code readability, etc.

We perform our audit according to the following procedure:

- automated analysis
  - we scan project's smart contracts with our own Solidity static code analyzer SmartCheck
  - we scan project's smart contracts with several publicly available automated Solidity analysis tools such as Remix, Ethlint, Solhint
  - we manually verify (reject or confirm) all the issues found by tools
- manual audit
  - we manually analyze smart contracts for security vulnerabilities
  - we check smart contracts logic and compare it with the one described in the documentation
  - we run tests and check code coverage
- report
  - we reflect all the gathered information in the report

Checked vulnerabilities

We have scanned Orbs smart contracts for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered (the full list includes them but is not limited to them):

- Reentrancy
- Front running
- DoS with (unexpected) revert
- DoS with block gas limit
- Gas limit and loops
- Locked money
- Integer overflow/underflow
- Unchecked external call
- ERC20 Standard violation
- Authentication with tx.origin
- Unsafe use of timestamp
- Using blockhash for randomness
- Balance equality
- Unsafe transfer of ether
- Fallback abuse
- Using inline assembly
- Short address attack
- Private modifier
- Compiler version not fixed
- Style guide violation
- Unsafe type deduction
- Implicit visibility level
- Use delete for arrays
- Byte array
- Incorrect use of assert/require
- Using deprecated constructions

Project overview

Project description

In our analysis we consider Orbs specification (”CONTRACT.md” in the repo) and smart contracts’ code (version on commit 803d630dc802396cd1369424b8810e00cc3f9f8c).

The latest version of the code

After the initial audit, some fixes were applied and the code was updated to the latest version (version on commit 92abd6b1dea152f958f1fc90afd03849e9f2f674). Also, the specification was updated (”CONTRACT.md” in the repo).

Project architecture

For the audit, we were provided with the truffle project. The project is an npm package and includes tests.

- The project successfully compiles with `truffle compile` command (see Compilation output in Appendix)
- The project successfully passes all the tests with 100% coverage

The total LOC of audited Solidity sources is 282.

Automated analysis

We used several publicly available automated Solidity analysis tools. Here are the combined results of SmartCheck, Solhint, and Ethlint scanning. All the issues found by tools were manually checked (rejected or confirmed).

True positives are constructions that were discovered by the tools as vulnerabilities and can actually be exploited by attackers or lead to incorrect contracts operation.

False positives are constructions that were discovered by the tools as vulnerabilities but do not consist a security threat.

Cases when these issues lead to actual bugs or vulnerabilities are described in the next section.

Tool	Rule	True positives	False positives
SmartCheck	Costly loop	2	3
	Use of SafeMath		1
	Extra gas consumption	2	
	Unsafe array's length manipulation	1	
Total SmartCheck		5	4
Solhint	Compiler version 0.4.26 does not satisfy the 0.5.10 semver requirement		3
	Avoid to make time-based decisions in your business logic		3
Total Solhint		0	6
Ethlint	Avoid using 'now' (alias to 'block.timestamp').		3
Total Ethlint		0	3
Total Overall		**5**	**13**

Manual analysis

The contracts were completely manually analyzed, their logic was checked and compared with the one described in the documentation. 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 smart contracts operation in current implementation. We highly recommend addressing them.

The audit showed no medium severity issues.

Low severity issues

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

CEI pattern violation

StakingContract.sol, line 309:

```
require(token.transferFrom(msg.sender, address(this),
_totalAmount), "StakingContract::distributeRewards -
insufficient allowance");
```

There are a few state variables written after the external call (tokens' transfer). So this statement violates CEI pattern. We recommend moving tokens' transfer to the end of function where all the calculations are done.

Comment from the developers: "We maintained the original code related to the CEI pattern item in StakingContract.sol, distributeRewards() as is as it makes the code more readable (first transferring the funds to the staking contract then distribute them), there's no reentrancy risk as the ORBS ERC20 is a standard ERC20."

Redundant code

There are a few duplicate getters:

- **StakingContract.sol**, lines 336–338:

```
function getStakeBalanceOf(address _stakeOwner) external
view returns (uint256) {
    return stakes[_stakeOwner].amount;
}
```

  At the same time `stakes` variable has `public` visibility level and a default getter as result.

- **StakingContract.sol**, lines 342–344:

```
function getTotalStakedTokens() external view returns
(uint256) {
    return totalStakedTokens;
}
```

  At the same time `totalStakedTokens` variable has `public` visibility level and a default getter as result.

- **StakingContract.sol**, lines 359–361:

```
function getToken() external view returns (IERC20) {
    return token;
}
```

  At the same time `token` variable has `public` visibility level and a default getter as result.

We highly recommend changing variables' visibility levels to `internal` in order to remove redundant getters, improve code readability and transparency and decrease cost of deployment.

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

Extra gas consumption

The following variables are read from the storage while there are equal memory variables:

- **StakingContract.sol**, line 124

```
emit MigrationManagerUpdated(migrationManager);
```

  There is equal memory variable `_newMigrationManager`.

- **StakingContract.sol**, line 136

```
emit EmergencyManagerUpdated(emergencyManager);
```

  There is equal memory variable `_newEmergencyManager`.

Reading from local memory requires significantly less gas compared to reading from the storage. Thus, we recommend using memory variables in order to reduce gas consumption.

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

The following variables are read from the storage on every iteration of the corresponding loops:

- **StakingContract.sol** line 148

```
for (uint i = 0; i < approvedStakingContracts.length; ++i)
```

- **StakingContract.sol**, line 168

```
while (i < approvedStakingContracts.length - 1)
```

Reading from local memory requires significantly less gas compared to reading from the storage. Thus, we recommend placing these variables into local memory in order to reduce gas consumption.

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

Code logic

StakingContract.sol, lines 168-174:

```
while (i < approvedStakingContracts.length - 1) {
    approvedStakingContracts[i] = approvedStakingContracts[i
+ 1];
    i++;
}
delete approvedStakingContracts[i];
approvedStakingContracts.length--;
```

This algorithm is not gas efficient. Also, `pop()` array method is more safe way of removing arrays' elements. We recommend using the following algorithm: swap the required element with the last one, delete the last element using `.pop()` method.

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

Compiler version

All Solidity files in the project start with

```
pragma solidity 0.4.26;
```

The most recent version as of the time of this writing is 0.5.13. We recommend using the latest compiler version, as newer versions include patches for recently discovered security vulnerabilities.

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

Notes

Gas limit and loops

The following loops traverse through arrays of variable length:

1. **StakingContract.sol**, line 313

```
for (uint i = 0; i < stakeOwnersLength; ++i)
```

2. **StakingContract.sol**, line 381

```
for (uint i = 0; i < stakeOwnersLength; ++i)
```

The traversed arrays are passed as functions parameters. Therefore, if there are too many items in these arrays, the execution of the corresponding functions will fail due to an out-of-gas exception.

In these cases, we recommend separating the calls into several transactions.

This analysis was performed by SmartDec.

Igor Sobolev, Security Analyst
Alexander Drygin, Security Analyst
Alexander Seleznev, Chief Business Development Officer

February 27, 2020

Appendix

Code coverage

```
---------------------------------|---------|----------|---------|---------|----------------|
File                             | % Stmts | % Branch | % Funcs | % Lines |Uncovered Lines |
---------------------------------|---------|----------|---------|---------|----------------|
contracts/                       |     100 |      100 |     100 |     100 |                |
IMigratableStakingContract.sol   |     100 |      100 |     100 |     100 |                |
IStakingContract.sol             |     100 |      100 |     100 |     100 |                |
StakingContract.sol              |     100 |      100 |     100 |     100 |                |
---------------------------------|---------|----------|---------|---------|----------------|
All files                        |     100 |      100 |     100 |     100 |                |
---------------------------------|---------|----------|---------|---------|----------------|
```

Compilation output

```
Compiling your contracts...
    ===========================
    > Compiling ./contracts/IMigratableStakingContract.sol
    > Compiling ./contracts/IStakingContract.sol
    > Compiling ./contracts/StakingContract.sol
    > Compiling ./contracts/tests/StakingContractWrapper.sol
    > Compiling ./contracts/tests/TestERC20.sol
    > Compiling openzeppelin-solidity/contracts/math/SafeMat
h.sol
    > Compiling openzeppelin-solidity/contracts/token/ERC20/
ERC20.sol
    > Compiling openzeppelin-solidity/contracts/token/ERC20/
IERC20.sol
    > Artifacts written to ./build/contracts
    > Compiled successfully using:
    - solc: 0.4.26+commit.4563c3fc.Emscripten.clang
```

Tests output

```
  Contract: StakingContract
    construction
     should not allow to create with a 0 cooldown
     should not allow to create with a 0 migration manager
     should not allow to create with a 0 emergency manager
     should not allow to create with a 0 token address
     should report version
     should correctly initialize fields
    setting of the migration manager
    as a regular account
     should not allow to set
    as a migration manager
     should set to a new address
     should not allow changing to 0
     should not allow changing to the same address
    setting of the emergency manager
    as a regular account
     should not allow to set
    as an emergency manager
     should set to a new address
     should not allow changing to 0
     should not allow changing to the same address
    adding/removal of migration destinations
    as a regular account
     should not allow adding a new contract
     should not allow removing of any contract
    as a migration manager
     should add new staking contracts
     should not allow adding a 0 address
     should not allow adding a duplicate contract
     should not allow adding more than 10 contracts
     should not allow adding again a previously removed cont
ract
     should remove contracts
     should not allow adding a 0 address
     should revert when trying to remove a non-existing cont
ract
    staking
    without a stake
     should allow staking
    accepting migration
     should allow staking on behalf of a different staker
     should not allow staking more tokens than the staker ha
```

```
s
     should not allow staking 0 tokens
     should not allow staking more tokens than the staker ha
s on behalf of a different staker
     should not allow staking on behalf of a 0 address
    distributing rewards
     should allow staking on behalf of different stakers in
batch
     should fail batch staking if token balance is insuffici
ent
     should fail batch staking if total batch amount is inco
rrect
     should fail batch staking if stake owners and amounts l
ists are in different sizes
     should fail batch staking if total token amount is 0
     should fail batch staking if one of the stake owners is
a 0 address
     should fail batch staking if one of the stake amounts i
s a 0
     should fail batch staking if called with empty lists
     should fail batch staking if unable to transfer
    with a stake
     should allow staking more tokens
    with unstaked tokens
     should allow staking
    with pending withdrawal
     should allow staking
    after full withdrawal
     should allow staking
    when stopped accepting new stake
     should not allow staking tokens
     should not allow staking tokens on behalf of a differen
t staker
     should not allow staking on behalf of different stakers
in batch
    when released all stake
     should not allow staking tokens
     should not allow staking tokens on behalf of a differen
t staker
     should not allow staking on behalf of different stakers
in batch
    unstaking
    without a stake
     should not allow unstaking
    with a stake
```

```
      should allow partially unstaking of tokens
      should allow unstaking of all tokens
      should not allow unstaking of 0 tokens
      should not allow unstaking more tokens than the staker
has staked
    with a pending withdrawal
      should not allow unstaking of more tokens
    after a full withdrawal
      should allow unstaking of more tokens
    when stopped accepting new stake
      should allow unstaking
    when released all stake
      should allow unstaking
    withdrawal
    without a stake
      should not allow withdrawal
    when released all stake
      should not allow withdrawal
    with a stake
      should not allow withdrawal
    with an unstaked stake
      should not allow withdrawal
    with a pending withdrawal
      should allow withdrawal of all unstaked tokens
      should not allow withdrawal if unable to transfer
    after full withdrawal
      should not allow withdrawal
    when stopped accepting new stake
      should allow withdrawal
    when released all stake
      should allow withdrawal of all unstaked tokens
    restaking
    without a stake
      should not allow restaking
    with a stake
      should not allow restaking
    with an unstaked stake
      should allow restaking
    pending withdrawal
      should allow restaking
    fully withdrawn
      should not allow restaking
    stopped accepting new stake
      should not allow restaking
    released all stake
```

```
      should not allow restaking
    migration to new staking contracts
    without a stake
      should not allow migration
    with a stake
      should allow migration
      should allow partial migration
      should only allow migration to an approved migration de
stination
      should not allow migration to a staking contract with a
different token
      should not allow migration if unable to approve
      should not allow migration of 0 tokens
      should not allow migration of more than staked tokens
    with an unstaked stake
      should only migrate tokens not in cooldown
    with a pending withdrawal
      should only migrate tokens not in cooldown
    after a full withdrawal
      should only migrate tokens not in cooldown
    when stopped accepting new stake
      should allow migration
    when released all stake
      should not allow migration
    emergency operations
    as a regular account
      should not allow requesting to stop accepting new stake
      should not allow requesting to release all stakes
      should not allow batch withdrawal of all stakes
    as an emergency manager
    when stopped accepting new stake
      should stop accepting new stakes
      should not allow requesting to stop accepting new stake
s again
      should allow requesting to release all stakes
    with an unstaked tokens
    when released all stake
      should allow withdrawal of all staked and unstaked toke
ns
      should not allow requesting to stop accepting new stake
      should not allow requesting again to release all stakes
    batch withdraw
    with an unstaked stakes
      should not allow batch withdrawal of all stakes
    when released all stake
```

```
     should allow batch withdrawal of all stakes
     should revert when trying to withdraw for a 0 address
     should revert when trying to withdraw for an address wi
thout any stake


    100 passing (1m)
```

Ethlint output

```
contracts/StakingContract.sol
    8:0       warning    Contract 'StakingContract' must be
preceded by 2 blank lines.
    103:4     warning    In case of more than 3 parameters,
drop each into its own line.
    119:16    error      Only use indent of 12 spaces.
    131:16    error      Only use indent of 12 spaces.
    142:16    error      Only use indent of 12 spaces.
    144:16    error      Only use indent of 12 spaces.
    149:20    error      Only use indent of 16 spaces.
    161:16    error      Only use indent of 12 spaces.
    206:16    error      Only use indent of 12 spaces.
    206:57    warning    Avoid using 'now' (alias to 'block.
timestamp').
    213:36    warning    Avoid using 'now' (alias to 'block.
timestamp').
    263:34    error      Opening brace must be on the line a
fter last modifier.
    264:16    error      Only use indent of 12 spaces.
    279:16    error      Only use indent of 12 spaces.
    281:16    error      Only use indent of 12 spaces.
    297:35    error      Opening brace must be on the line a
fter last modifier.
    303:16    error      Only use indent of 12 spaces.
    305:16    error      Only use indent of 12 spaces.
    309:16    error      Only use indent of 12 spaces.
    351:33    error      Opening brace must be on the line a
fter returns declaration.
    414:16    error      Only use indent of 12 spaces.
    433:49    warning    Avoid using 'now' (alias to 'block.
timestamp').
    458:34    error      Opening brace must be on the line a
```

```
fter returns declaration.

    contracts/tests/StakingContractWrapper.sol
    6:0      warning    Contract 'StakingContractWrapper' mu
st be preceded by 2 blank lines.
    8:114    warning    Code contains empty block

    contracts/tests/TestERC20.sol
    5:0    warning    Contract 'TestERC20' must be preceded
by 2 blank lines.

     18 errors, 8 warnings found.
```