SmartDec

DCorp Smart Contracts Security Analysis

This report is private.

Published: August 5, 2019.

Abstract

In this report, we consider the security of the DCorp 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 DCorp 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. They do not endanger project security. Nevertheless, we recommend addressing them.

General recommendations

The contracts code is of high code quality. The audit did not reveal any issues that endanger project security. However, if the developers decide to improve the code, we recommend fixing the Code coverage issue, writing the Documentation and removing Redundant code.

However, mentioned above are minor issues. They do not affect code operation.

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 other kinds of bugs. ✓

Compliance with the documentation

There is no documentation provided with the code. ✓

Tests

All the tests passed successfully. However, the code coverage was not generated. See Code coverage issue. ✓

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 Slither, and Solhint
  - we manually verify (reject or confirm) all the issues found by tools
- manual audit
  - we manually analyze smart contracts for security vulnerabilities
  - we run tests
- report
  - we report all the issues found to the developer during the audit process
  - we check the issues fixed by the developer
  - we reflect all the gathered information in the report

Checked vulnerabilities

We have scanned DCorp 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 DCorp smart contracts' code (”proposalconfidence.zip”, sha1sum: 1d84643ef328c04578b0ead157b65908ecc8b979, commit: 953b83257894082ad45efc8579bcb68e1ff34b50).

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. However, the code coverage was not generated (see Code coverage issue)

The total LOC of audited Solidity sources is 454.

Automated analysis

We used several publicly available automated Solidity analysis tools. Here are the combined results of SmartCheck, Solhint, and Remix 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
Slither	External function	9	
	Naming convention	11	
	Unused return	1	
	Solc version		1
Total Slither		21	1
SmartCheck	Compiler version not fixed	1	
	Prefer external to public visibility level	10	
	Private modifier		3
Total SmartCheck		11	3
Solhint	Avoid to make time-based decisions in your business logic		7
	Compiler version must be fixed	1	
	Modifier name must be in mixedCase	10	
Total Solhint		11	7

Total Overall	**43**	**11**

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 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.

Code coverage

The provided code contains tests, however the code coverage generation fails.

We highly recommend not only covering the code with tests but also making sure that the test coverage is sufficient.

Redundant code

`only_after()` modifier at **DcorpDissolvementProposal.sol**, line 90 is redundant:

```
modifier only_after(uint _timestamp)
```

We highly recommend removing redundant code in order to improve code readability and transparency and decrease the cost of deployment and execution.

Lack of documentation

There is no documentation provided with the code.

We recommend writing the documentation in order to make the code easier to read, maintain and verify.

External function visibility

In **DcorpDissolvementProposal.sol** the following functions should have `external` visibility:

- `isDeploying()`
- `isDeployed()`
- `isExecuted()`
- `deploy()`
- `getTotalSupply()`
- `balanceOf()`
- `execute()`
- `withdraw()`
- `retrieveEther()`

We recommend changing visibility level to `external` in order to increase code readability and decrease gas costs.

Naming convention

According to Solidity documentation, modifier names should be written in mixedCase. However, lower case with underscores are used for modifiers in **DcorpDissolvementProposal.sol**.

We recommend following naming convention since this improves code readability.

Unchecked return value

`execute()` function in **DcorpDissolvementProposal.sol** ignores return values of external calls at lines 316 and 317:

```
IManagedToken(address(drpsToken)).lock();
IManagedToken(address(drpuToken)).lock();
```

We recommend checking whether tokens were locked by making external calls inside `require()` function.

Compiler version

Solidity source files indicate the versions of the compiler they can be compiled with.

```
Example:
pragma solidity ^0.5.8; // bad: compiles w 0.5.8 and above
pragma solidity 0.5.8; // good: compiles w 0.5.8 only
```

We recommend following the latter example, as future compiler versions may handle certain language constructions in a way the developers have not foreseen. Besides, we recommend using the latest compiler version – 0.5.10 at the moment.

This analysis was performed by SmartDec.

Alexander Seleznev, Chief Business Development Officer
Boris Nikashin, Project Manager
Alexander Drygin, Analyst
Pavel Kondratenkov, Analyst

August 5, 2019

Appendix

Compilation output

```
Compiling your contracts...
===========================
> Compiling ./contracts/Migrations.sol
> Compiling ./contracts/infrastructure/authentication/IAuthe
nticator.sol
> Compiling ./contracts/infrastructure/behaviour/IObservable
.sol
> Compiling ./contracts/infrastructure/behaviour/Observable.
sol
> Compiling ./contracts/infrastructure/modifier/InputValidat
or.sol
> Compiling ./contracts/infrastructure/ownership/IMultiOwned
.sol
> Compiling ./contracts/infrastructure/ownership/IOwnership.
sol
> Compiling ./contracts/infrastructure/ownership/ITransferab
leOwnership.sol
> Compiling ./contracts/infrastructure/ownership/MultiOwned.
sol
> Compiling ./contracts/infrastructure/ownership/Ownership.s
ol
> Compiling ./contracts/infrastructure/ownership/Transferabl
eOwnership.sol
> Compiling ./contracts/source/DcorpDissolvementProposal.sol
> Compiling ./contracts/source/token/IManagedToken.sol
> Compiling ./contracts/source/token/IToken.sol
> Compiling ./contracts/source/token/ManagedToken.sol
> Compiling ./contracts/source/token/Token.sol
> Compiling ./contracts/source/token/observer/ITokenObserver
.sol
> Compiling ./contracts/source/token/observer/TokenObserver.
sol
> Compiling ./contracts/source/token/retriever/ITokenRetriev
er.sol
> Compiling ./contracts/source/token/retriever/TokenRetrieve
r.sol
> Compiling ./contracts/test/Accounts.sol
> Compiling ./contracts/test/mock/MockDRPSToken.sol
> Compiling ./contracts/test/mock/MockDRPUToken.sol
> Compiling ./contracts/test/mock/MockDcorpProxy.sol
> Compiling ./contracts/test/mock/MockToken.sol
> Compiling ./contracts/test/mock/MockTokenObserver.sol
> Compiling ./contracts/test/mock/MockWhitelist.sol
> Compiling ./contracts/test/proxy/CallProxy.sol
> Compiling ./contracts/test/proxy/CallProxyFactory.sol
> Artifacts written to /home/drygin/audit/proposalconfidence
/build/contracts
> Compiled successfully using:
   - solc: 0.5.8+commit.23d335f2.Emscripten.clang
```

Tests output

```
  Contract: Dissolvement Proposal (Deploy)
     Should be in the deploying stage initially
     Should not be able to deploy before receiving eth (125m
s)
     Should not accept eth from any other address than the p
rev proxy (121ms)
     Should be able to send eth in the deploying stage (159m
s)
     Should be able to deploy after receiving eth (173ms)
     Should not be able to send eth to the dissolvement prop
osal in the deployed stage (44ms)
     Should not allow any account other than the drp tokens
to call notifyTokensReceived() (74ms)
     Should accept drps tokens in the deployed stage (99ms)
     Should accept drpu tokens in the deployed stage (94ms)

  Contract: Dissolvement Proposal (Execution)
     Should accept and record 18000 drps tokens in the deplo
yed state (86ms)
     Should accept and record 540000 drps tokens in the depl
oyed state (81ms)
     Should accept and record 400 drps tokens in the deploye
d state (89ms)
     Should accept and record 18000 drpu tokens in the deplo
yed state (125ms)
     Should accept and record 540000 drpu tokens in the depl
oyed state (83ms)
     Should accept and record 600 drpu tokens in the deploye
d state (87ms)
```

```
     Should not be able to execute before the claim period h
as ended (68ms)
     Should not accept and record drps tokens after the clai
ming period (129ms)
     Should not accept and record drpu tokens after the clai
ming period (102ms)
     Should be able to execute after the claim period has en
ded (105ms)
     Should have a claimable balance
     Drps token should be locked after executing the proposa
l
     Drpu token should be locked after executing the proposa
l

  Contract: Dissolvement Proposal (Execution)
     DRPS account 0x0c807ABC20eAA87848852D2aa68A54f438324db9
should not be able to withdraw when not authenticated (71ms)
     DRPS account 0x0e1BAd461D11Ef4B2FfC0958223B3f59193c78Db
should not be able to withdraw when not authenticated (81ms)
     DRPS account 0xc58D5c693E660bD8eC0769251C519265bD919947
should not be able to withdraw when not authenticated (89ms)
     DRPU account 0x7ABE98044ba7E23c072A150A40FBC852e22C8b0b
should not be able to withdraw when not authenticated (83ms)
     DRPU account 0x6baeC578213f2466D5d7d263E33692C371D0ef0A
should not be able to withdraw when not authenticated (90ms)
     DRPU account 0xeB3A3Dc8Ddc87CAAaCde75823dE19dC2fe2a56C1
should not be able to withdraw when not authenticated (86ms)
     DRPS account 0x0c807ABC20eAA87848852D2aa68A54f438324db9
should be able to withdraw when authenticated (101ms)
     DRPS account 0x0e1BAd461D11Ef4B2FfC0958223B3f59193c78Db
should be able to withdraw when authenticated (107ms)
     DRPS account 0xc58D5c693E660bD8eC0769251C519265bD919947
should be able to withdraw when authenticated (113ms)
     DRPU account 0x7ABE98044ba7E23c072A150A40FBC852e22C8b0b
should be able to withdraw when authenticated (100ms)
     DRPU account 0x6baeC578213f2466D5d7d263E33692C371D0ef0A
should be able to withdraw when authenticated (100ms)
     DRPU account 0xeB3A3Dc8Ddc87CAAaCde75823dE19dC2fe2a56C1
should be able to withdraw when authenticated (91ms)
     Should not allow withdraws after the withdraw period (1
32ms)
     Should allow the owner to destroy the contract after th
e withdraw period (70ms)
```

```
   36 passing (5s)
```