Quantstamp Smart Contract Audit

Disciplina Token Contract Audit

This smart contract audit was prepared by Quantstamp, the protocol for securing smart contracts.

This security audit report follows a generic template. Future Quantstamp reports will follow a similar template and they will be fully generated by automated tools.

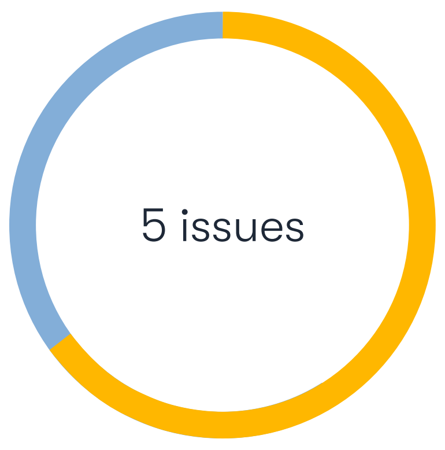
Quantstamp helps to secure blockchain applications such as smart contracts. We are developing a new protocol for smart contract verification, performing professional audits and consultations, and developing security tools. Quantstamp also has expertise in application security and secure software development.

Executive Summary

Type	ERC20-based token contract
Consultants	3
Timeline	5 days
Language	Solidity + JavaScript
Methods	Architecture Review, Functional Testing, Computer-aided Verification, Manual Review
Specification	Our understanding of the specification was based on the following documentation: • Disciplina Whitepaper • DISCIPLINA blockchain platform: Monetary policy We also elicited some of the implicit requirements from Disciplina team through private communication channels.
Source Code	The following source code was reviewed during the audit:

Repository	Commit
contracts	ed864b7

Total Issues	**5**
High Risk Issues	0
Medium Risk Issues	0
Low Risk Issues	2
Informal Risk Issues	3



Severity Categories	
Informal	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
Low	The risk is relatively small or is not a risk the client has indicated is important.
Medium	Individual user's information is at risk, exploitation would be detrimental for the client's reputation, moderate financial impact.
High	Large numbers of users impacted, catastrophic for client's reputation, or serious financial implications.

Overall Assessment

The Disciplina token makes heavy use of pre-existing library contracts, specifically from OpenZeppelin. As Disciplina token is ERC20-compatible, it does exhibit the "standard" ERC20 race condition between `approve` and `transferFrom` (mitigated by `increase` / `decreaseApproval`).

Furthermore, it features the centralization of power. While not a vulnerability, if the contract owner's private key is compromised, then the following issues may arise:

- arbitrary token minting,
- adding/removing minting allowance to/from arbitrary addresses,
- finishing the minting process prematurely.

As the Disciplina team explained, they favor this design due to its flexibility. As the token contract is meant for the pre-sale stage only, this centralization of power is viewed as a temporary aspect and bears low risk of attack.

Beyond those mentioned above, Quantstamp had no additional findings of potential vulnerabilities at the time of analysis.

Methodology

The review was conducted during 2018-June-22 through 2018-June-26 by the Quantstamp team, which included senior engineers Alex Murashkin, Martin Derka, and Kacper Bak.

Their procedure can be summarized as follows:

1. Code review
    a. Review of the specification
    b. Manual review of code
    c. Comparison to specification
2. Testing and automated analysis
    a. Test coverage analysis
    b. Symbolic execution (automated code path evaluation)
3. Best-practices review
4. Itemize recommendations

Security Audit

Quantstamp's objective was to evaluate the Disciplina ERC20 based contract repository for security-related issues, code quality, and adherence to best-practices.

Possible issues include (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights

Toolset

The below notes outline the setup and steps that were performed.

Testing setup:

- Truffle v4.1.8
- Ganache v1.1.0
- solidity-coverage v0.5.0
- Oyente v0.2.7
- Mythril v0.17.9

Steps taken to run the full test suite:

- Installed the solidity-coverage tool: `npm install --save-dev solidity-coverage`.
- Ran the coverage tool: `./node_modules/.bin/solidity-coverage`.
- Installed the mythril tool from Pypi: `pip3 install mythril`.
- Ran the `mythril` tool: `myth -x /contracts/truffle/contracts/`.
- Installed the `Oyente` tool from Docker: `docker pull luongnguyen/oyente && docker run -i -t luongnguyen/oyente`.
- Ran the `Oyente` tool: `cd /oyente/oyente && python oyente.py -s Contract.sol`.

Evaluation

Code Coverage

The file DisciplinaToken.sol features a 69.35% statement code coverage with 70.77% line coverage. This is due to the embedded SafeMath contracts that are not tested. The DisciplinaToken contract itself starts on line 247 and appears to be completely covered by tests. As the SafeMath libraries are currently embedded using clone-and-own approach, Quantstamp recommends that they are tested as well.

```
./node_modules/.bin/solidity-coverage
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	**69.35**	**50**	**72.73**	**70.77**	
DisciplinaToken.sol	69.35	50	72.73	70.77	...237,238,239
All files	**69.35**	**50**	**72.73**	**70.77**	

Clone-and-Own Approach to Using External Libraries

A comment in the code states that another token contract was used as a source. From the development perspective, it is beneficial as it reduces the amount of effort. However, from the security perspective, it involves some risks as the source may not follow the best practices, may contain a security vulnerability, or may include intentionally or unintentionally modified upstream libraries.

In this case, it appears that the source embeds contract interfaces and methods from the OpenZeppelin library, and the Disciplina token contract inherently benefits from it. However, as opposed to the clone-and-own approach, a good industry practice is using the Truffle framework for managing library dependencies. This eliminates the risk of the clone-and-own based approaches yet allows for following best practices, such as, using libraries.

Allowance Double Spend Exploit

As it presently is constructed, the contract is vulnerable to the allowance double-spend exploit, similarly to other ERC20 tokens.

The exploit (as described below) is mitigated through use of functions that increase/decrease the allowance relative to its current value, such as `increaseApproval` and `decreaseApproval`.

The following is a description of the exploit:

1. Alice allows Bob to transfer `N` amount of Alice's tokens `(N>0)` by calling the `approve` method on `Token` smart contract (passing Bob's address and N as method arguments)
2. After some time, Alice decides to change from `N` to `M` `(M>0)` the number of Alice's tokens Bob is allowed to transfer, so she calls the `approve` method again, this time passing Bob's address and `M` as method arguments
3. Bob notices Alice's second transaction before it was mined and quickly sends another transaction that calls the `transferFrom` method to transfer N Alice's tokens somewhere
4. If Bob's transaction will be executed before Alice's transaction, then Bob will successfully transfer N Alice's tokens and will gain an ability to transfer another `M` tokens
5. Before Alice notices any irregularities, Bob calls transferFrom method again, this time to transfer `M` Alice's tokens.

Allowance Double Spend Exploit

Ultimately, Alice's attempt to change Bob's allowance from `N` to `M (N>0 and M>0)` made it possible for Bob to transfer `N+M` of Alice's tokens, despite Alice's intention of not allowing this amount.

Pending community agreement on an ERC standard that would protect against this exploit, we recommend that developers of applications dependent on `approve` / `transferFrom` should keep in mind that they have to set allowance to 0 first and verify if it was used before setting the new value. Teams who decide to wait for such a standard should make these recommendations to application developers who work with their token contract.

Allowing for Minting Both Old and New Amount in Case of Changing Allowance

Similar to the allowance double-spend exploit above, the contract allows an account for minting both old and new amount of tokens in an edge case scenario when the allowance is being changed simultaneously with minting the old allowance. According to the team, *"We have instructions that require the administrator to first zero the amount (if it is not already equal to zero), and then set the desired value. It is just an additional precaution against the attack similar to erc-20 allowance attack."*, which mitigates the issue.

Centralization of Power

The smart contract does not put a restriction on the amount of tokens the owner could authorize to mint. While the approximate supply cap is known, 95 000 000 DSCP, it is not enforced in the contract, thus contributors must trust that the owner will mint the predefined number of tokens. According to the team, the exact supply is not known due to variable bonus size at the private sale stage, and the supply limit is to be enforced in the crowdsale contract. In addition, a trusted token owner is one of the security assumptions made by the team.

Naming

Our recommendation is to keep function and event naming consistent. The `allowMint()` function emits the event `MintApproval`. In the same vein, we recommend renaming `MintFinished` to `MintingFinished`.

Adherence to Specification

With minimal written specification we were unable to judge to what degree the code conforms to the specification. Private conversation with the Disciplina team convinced us that the contract code implements the desired functionality within the context of its intended usage.

Extensive Test Coverage

The contract benefits from extensive test coverage within the Truffle project, checking for numerous security and logic flaws within.

Toolset Warnings

Symbolic execution (the Oyente tool) did not detect any vulnerabilities of types Parity Multisig Bug 2, Transaction-Ordering Dependence (TOD), Callstack Depth Attack, Timestamp Dependency, and Re-Entrancy Vulnerability.

Mythril tool has not detected any vulnerabilities of kinds Integer underflow, Unprotected functions, Missing check on `call` return value, Re-entrancy, Multiple sends in a single transaction, External call to untrusted contract, `delegatecall` or `callcode` to untrusted contract, Timestamp dependence, Use of `tx.origin`, Predictable RNG, Transaction order dependence, Use `require()` instead of `assert()`, Use of deprecated functions, Detect tautologies.

Code Documentation

We noted that a majority of the functions were self-explanatory, and standard documentation tags (such as `@dev`, `@param`, and `@returns`) were included.

Appendix

Truffle Test Results

Below are SHA256 file signatures of the relevant files reviewed in the audit.

```
$ shasum -a 256 ./contracts/*
e6c019c44873810de9cdc871f56178ccf2b951322179a70b2f86524dfb1e0414  ./contracts/DisciplinaToken.sol
1cb2333ba7589af0731b50589a691930343afa45ff23d0cd61c3e6317bd6c33b  ./contracts/Migrations.sol
```

Truffle Test Results

Contract: DisciplinaToken

```
  after token creation
    ✓ sender should be token owner
  minting finished
    when the token minting is not finished
      ✓ should return false
    when the token minting is finished
      ✓ should return true
  finish minting
    when the sender is the token owner
      when the token minting is not finished
        ✓ should finish token minting (48ms)
        ✓ should emit a mint finished event
      when the token minting is finished
        ✓ should revert the transaction
    when the sender is not the token owner
      ✓ should revert the transaction
  allow mint
    when the sender is the token owner
      when the token minting is not finished
        ✓ should emit a minting approval event (38ms)
        ✓ should set the minting allowance of the minter (68ms)
        if called multiple times
          ✓ should set the minting allowance of the minter (89ms)
      when the token minting is finished
        ✓ should revert the transaction
    when the sender is not the token owner
      ✓ should revert the transaction
  mint
    when minter mints tokens
      less than or equal to his minting allowance
        when the minting is not finished
          ✓ should log minting event
          ✓ should log transfer event
          ✓ should increase total supply (55ms)
          ✓ should increase the balance of the beneficiary (52ms)
          ✓ should decrease the minting allowance of the minter
        when the minting is finished
          ✓ should revert the transaction
      more than his minting allowance
        ✓ should revert the transaction
  transfers
    transfer
      when the minting is finished
        ✓ should transfer tokens (43ms)
      when the minting is not finished
        ✓ should revert the transaction
    transferFrom
      when the minting is finished
        ✓ should transfer tokens (80ms)
      when the minting is not finished
        ✓ should revert the transaction
```

Disclosure

Purpose of Report

The scope of our review is limited to a review of Solidity code and only the source code we note as being within the scope of our review within this report. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks.

The report is not an endorsement or indictment of any particular project or team, and the report does not guarantee the security of any particular project. This report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset.

No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project.

Disclaimer

While Quantstamp delivers helpful but not-yet-perfect results, our contract reports should be considered as one element in a more complete security analysis. A warning in a contract report indicates a potential vulnerability, not that a vulnerability is proven to exist.

Timeliness of Content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by QTI; however, QTI does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Links to Other Websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp Technologies Inc. (QTI). Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that QTI are not responsible for the content or operation of such web sites, and that QTI shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that QTI endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. QTI assumes no responsibility for the use of third-party software on the website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

Notice of Confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These material are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.