

Blockchain Capital Token Audit

APRIL 27, 2017 | IN SECURITY AUDITS | BY OPENZEPPELIN SECURITY



The [Blockchain Capital Team](#) asked us to review and audit [their new BCAP Token](#) contract code. We looked at their code and now publish our results.

The audited contract is at [their BCAPToken GitHub repo](#). The version used for this report is commit `5cb5e76338cc47343ba9268663a915337c8b268e`, corresponding to version 0.7. The main contract file is [BCAPToken.sol](#).

Code quality is very good, and well commented and modularized. This is one of the best written projects we audited so far.

Update: the Blockchain Capital team fixed some of our recommendations [in their latest version](#).

Here's our assessment and recommendations, in order of importance:

Severe

We haven't found any severe security problems with the code.

Potential problems

Use latest version of Solidity

[Current code](#) is written for old versions of solc (0.4.1). With [the storage overwriting vulnerability](#) found on versions of solc prior to 0.4.4, there's a risk this code can be compiled with vulnerable versions. We recommend changing the solidity version pragma for the latest version (`pragma solidity ^0.4.10;`) to enforce latest compiler version to be used.

EDIT: The Argon Group replied: "The pragma directive assures that no language constructions introduced after 0.4.1 are used—for the purpose of language maturity. It also allows compiling by a version up to 0.5.0 (not yet released). The current binaries are compiled with version 0.4.10. Therefore there is no risk of using a too old compiler version for this particular contract."

Key escrow service for critical accounts

The `tokenIssuer` address state variable in [line 49 of StandardToken](#) has full control of the token supply. Furthermore, it's the same address that can freeze and unfreeze transactions in [BCAPToken](#). The corresponding private key should have

really high security standards. Consider using a multisig wallet for that address, and a [key escrow service](#) to further protect it.

Warnings

Bug Bounty

Formal security audits are not enough to be safe. We recommend implementing an [automated contract-based bug bounty](#) and setting a period of time where security researchers from around the globe can try to break the contract's invariants. For more info on how to implement automated bug bounties with OpenZeppelin, [see this guide](#).

Avoid duplicated code

Duplicate code makes it harder to understand the code's intention and thus, auditing the code correctly. It also increases the risk of introducing hidden bugs when modifying one of the copies of some code and not the others.

The logic of the contracts `Token`, `AbstractToken`, and `SafeMath` are very similar to OpenZeppelin library contracts `ERC20Token`, `StandardToken`, and `SafeMath`, respectively. Consider avoiding code repetition, which can bring regression problems and [introduce unexpected bugs](#). Consider using the standard modules from OpenZeppelin.

Another instance of duplicated code is [the owner variable of BCAPToken.sol](#). Consider using OpenZeppelin's `Ownable` as an equivalent. [Checks for owner being the msg.sender](#) are implemented in `Ownable` too, but could also be extracted as a function modifier.

Fail early and loudly

A simple yet powerful programming good practice is to [make your code fail as promptly as possible](#). And be loud about it. We want to avoid a contract failing silently, or continuing execution in an unstable or inconsistent state. Consider changing all precondition checks to `throw`ing to follow [this good practice](#). Some parts of the code do this correctly, but we'd like to see more consistency on this pattern.

Some places where this could be improved are:

- <https://github.com/BCAPtoken/BCAPToken/blob/5cb5e76338cc47343ba9268663a915337c8b268e/sol/BCAPToken.sol#L30>
- <https://github.com/BCAPtoken/BCAPToken/blob/5cb5e76338cc47343ba9268663a915337c8b268e/sol/BCAPToken.sol#L42>

Additional Information and Notes

- BCAPToken is [ERC20 compliant](#), as [required by the functional specification](#).
- `totalSupply` from ERC20 standard is implemented, but it's not taking into account the `centralBank` account, which has the power to create tokens at will. This doesn't mean the ERC20 is not respected, but may be unexpected by some users.
- Similarly, `balanceOf` hides the balance of the `centralBank` account.
- `transfer main flow` is correctly implemented.
- `transfer exceptional flow #1` is correctly implemented.
- `transfer exceptional flow #2` is correctly implemented, but "maximum allowed number of tokens in circulation" is just `MAX_UINT256`. Maybe this should be a configuration parameter?
- `transfer exceptional flow #3` is correctly implemented.
- `transfer exceptional flow #4` is correctly implemented.
- `transfer exceptional flow #5` is correctly implemented.

- transfer exceptional flow #6 is correctly implemented.
- transfer exceptional flow #7 is correctly implemented.
- transfer exceptional flow #8 is correctly implemented.
- transfer exceptional flow #9 is correctly implemented.
- transferFrom main flow is correctly implemented.
- transferFrom exceptional flow #1 is correctly implemented.
- transferFrom exceptional flow #2 is correctly implemented.
- transferFrom exceptional flow #3 is correctly implemented.
- transferFrom exceptional flow #4 is correctly implemented.
- transferFrom exceptional flow #5 is correctly implemented.
- transferFrom exceptional flow #6 is correctly implemented.
- transferFrom exceptional flow #7 is correctly implemented.
- transferFrom exceptional flow #8 is correctly implemented.
- transferFrom exceptional flow #9 is correctly implemented.
- transferFrom exceptional flow #10 is correctly implemented.
- approve main flow is correctly implemented.
- allowance main flow is correctly implemented.
- Deploy main flow is correctly implemented.
- Freeze main flow is correctly implemented.
- Freeze exceptional flow #1 is correctly implemented.
- Freeze exceptional flow #2 is correctly implemented.
- Unfreeze main flow is correctly implemented.
- Unfreeze exceptional flow #1 is correctly implemented.
- Unfreeze exceptional flow #2 is correctly implemented.

- [BCAP Token limits](#) are correctly set.
- [Tests](#) are correctly implemented.
- No need to use `safeSub` in [line 32 of StandardToken.sol](#), as `MAX_UINT256` will always be greater or equal than any `uint256`.
- Good idea to [use events to get results of transactional functions in tests](#).

Conclusions

No severe security issues were found. Some changes were recommended to follow best practices and reduce potential attack surface.

Update: the Blockchain Capital team fixed some of our recommendations [in their latest version](#).

Code quality is very good, and well commented and modularized. This is one of the best written projects we audited so far.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the BCAP Token contract. We have not reviewed the related Blockchain Capital project. The above should not be construed as investment advice or an offering of tokens. For general information about smart contract security, check out our thoughts [here](#).

Security Audits

- If you are interested in smart contract security, you can continue the discussion in our [forum](#), or even better, [join the team](#) 🚀

- If you are building a project of your own and would like to request a security audit, please do so [here](#).

RELATED POSTS



Metal Token Audit

 OpenZeppelin | security

SECURITY AUDITS

Metal Token Audit

The Metal team asked us to review and audit their new Metal Token contract code. We looked at their...

[READ MORE](#)



Everus Token Audit

 OpenZeppelin | security

SECURITY AUDITS

Everus Token Audit

The Everus team asked us to review and audit their Everus Token (EVR) contract. We looked at the...

[READ MORE](#)



BitClave Token Audit

 OpenZeppelin | security

SECURITY AUDITS

BitClave Token Audit

The BitClave team asked us to review and audit their Consumer Activity Token (CAT) contracts. We...

[READ MORE](#)

OpenZeppelin

Email*

Get our monthly news roundup

由 reCAPTCHA 提供保护
隐私权 - 使用条款

SIGN UP

© OpenZeppelin 2017-2022

Privacy | Terms of Service

Products

Contracts

Defender

Security

Security Audits

Learn

Docs

Forum

Ethernaut

Company

Website

About

Jobs

Logo Kit