Infomatix

smart contracts final audit report

November 2021

hashex.org

contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the Infomatix team to perform an audit of their smart contracts. The audit was conducted between November 13 and November 16, 2021.

The code located in the Github repository @infomatixDev/Infomatix was audited after the commit 993879.

The whitepaper is available on the project's website.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts.
- Formally check the logic behind given smart contracts.

Information in this report should be used to understand the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

Update: the Infomatix team updated the whitepaper.

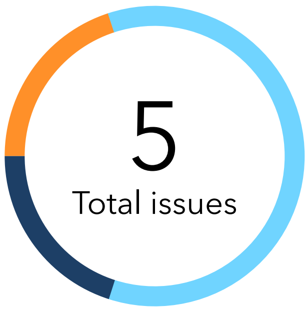
2.1 Summary

Project name	Infomatix
URL	https://infomatix.io/
Platform	Binance Smart Chain
Language	Solidity

2.2 Contracts

Name	Address
Infomatix	`0xdf727040d3997b5d95dee8c661fa96e3c13ee0c9`

3. Found issues



High		1 (20%)
Low		3 (60%)
Info		1 (20%)

C1. Infomatix

ID	Severity	Title	Status
C1-01	High	Token inflation mechanism starts after 1 year	Acknowledged
C1-02	Low	Pragma version is not fixed	Acknowledged
C1-03	Low	Unused code	Acknowledged
C1-04	Low	Proper use of external and public function modifiers	Acknowledged
C1-05	Info	Solidity naming convention violations	Acknowledged

4. Contracts

C1. Infomatix

Overview

A BEP20 token with an inflation mechanism. 3 years after the deployment the token will start to mint once a year 1% of the total supply to the owner.

Issues

C1-01 Token inflation mechanism starts after 1 year ● High ✓ Acknowledged

The if statement that checks when the token should start minting uses '=' instead of '=='. The first branch of the if statement that should start minting after 3 years will never be called and the token will start minting after 1 year of deployment.

```
    function _transfer(address sender, address recipient, uint256 amount) internal virtual
{
        ...
        if(_annualMintingStarted = false) {
            if(block.timestamp > _TokenDeployedDate.add(94670778)) {
                _mint(owner(), _totalSupply.div(100));
                _annualMintingStarted = true;
                _lastMintingEvent = block.timestamp;
            }
        } else {
            if(block.timestamp > _lastMintingEvent.add(31556926)){
                _mint(owner(), _totalSupply.div(100));
                _lastMintingEvent = block.timestamp;
            }
        }
    }
```

This issue does not anyhow affect the token's behavior other than changing the start day of minting and does not impose any additional risks.

Recommendation

The check should be:

```
if(!_annualMintingStarted)
```

Update

The Infomatix team updated the whitepaper to comply with deployed smart contract behavior (minting starts after 1 year).

Infomatix team response

On November 16th, 2021, the leadership team at Infomatix were made aware of a discrepancy between the minting functionality described in the whitepaper and the smart contract, as outlined by a commissioned HashEx audit. The whitepaper had stipulated that the 1% inflationary minting functionality would be triggered after a three-year period from token generation event, whereas the smart contract had this functionality reflected as commencing after a 12 month period from token generation event. Since this discrepancy was identified, the Infomatix whitepaper has been updated to reflect the correct inflationary mechanism of 12 months from token generation event, as is described in the smart contract, and existing investors notified of this update. Further treatment of this this mechanism, is outlined as originally described in the whitepaper.

C1-02 Pragma version is not fixed

Low | Acknowledged

The code can be compiled with a wide range of Solidity versions

```
pragma solidity ^0.8.0;
```

This can lead to discrepancies in the behavior of smart contracts compiled with a different Solidity version.

Recommendation

We recommend fixing pragma version to ensure that deployed contract will be compiled with the same Solidity version as it was tested:

```
pragma solidity 0.8.7;
```

C1-03 Unused code

Low Acknowledged

The functions `_burn()` and `_setupDecimals()` are not used anywhere and can be removed.

SafeMath library is imported but isn't used. As the contract specifies Solidity versions above 0.8 that have build-in safe math checks, this library can be removed.

Recommendation

We recommend removing the unused code to reduce the size of the bytecode of the compiled contract.

C1-04 Proper use of external and public function modifiers

Low Acknowledged

The functions `renounceOwnership()`, `transferOwnership()`, `name()`, `symbol()`, `decimals()`, `totalSupply()`, `balanceOf()`, `getOwner()`, `transfer()`, `transferFrom()`, `approve()`, `allowance()`, `increaseAllowance()`, `decreaseAllowance()` should be declared external. This will save gas on calling them.

Recommendation

Make these functions external instead of public.

C1-05 Solidity naming convention violations

Info Acknowledged

Variables `_lastMintingEvent`, `_TokenDeployedDate`, `_annualMintingStarted` should be named in camelCase according to the Solidity style guide.

5. Conclusion

One high severity and several low and informational severity were found. These issues do not seriously affect the token's behavior and do not impose any risks.

It must be noted that at the time of audit 83.7% of the token supply is held on one EOA address although the whitepaper describes the distribution between several accounts. Holding most of the token on one EOA address is an additional risk for the investors.

Update: team updated the whitepaper to conform to the current token behavior (one year delay before the start of the minting). The official team response can be seen below the issue description.

Update 2: 48% of the total supply was sent by the team to a vesting contract with equal amount releases every month for 9 months and 24.8% of token supply was sent to vesting contract with 3 months releases. It must be noted that the vesting contracts are out of the scope of the current audit.

Appendix A. Issues' severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

contact@hashex.org
@hashexbot
blog.hashex.org
linkedin
github
twitter

HashEx
BLOCKCHAIN SECURITY