This document may contain confidential information about IT systems and intellectual property of the customer as well as information about potential vulnerabilities and methods of their exploitation. The report containing confidential information can be used internally by the customer or it can be disclosed publicly after all vulnerabilities fixed – upon decision of customer.

### Document

<table>
<thead>
<tr>
<th>Name</th>
<th>Smart Contract Code Secondary Review and Security Analysis Report for MainCoin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform</td>
<td>Ethereum / Solidity</td>
</tr>
<tr>
<td>Link</td>
<td><a href="https://etherscan.io/address/0x9F0F1BE08591AB7D990FAF910B38ED560E4D5BF#code">https://etherscan.io/address/0x9F0F1BE08591AB7D990FAF910B38ED560E4D5BF#code</a></td>
</tr>
<tr>
<td>Date</td>
<td>29.03.2019</td>
</tr>
</tbody>
</table>
Table of contents

Document ................................................................. 2
Table of contents ............................................................. 3
Introduction ................................................................. 4
Scope ........................................................................... 4
Executive Summary ......................................................... 5
Severity Definitions ......................................................... 6
AS-IS overview .............................................................. 6
Audit overview ............................................................... 9
Conclusion ................................................................. 11
Disclaimers ................................................................. 12
Appendix A. Evidences .................................................... 13
Appendix B. Automated tools reports ................................. 14
Introduction

Hacken OÜ (Consultant) was contracted by MainCoin (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer’s smart contract and its code review conducted between March 27th, 2019 – March 29th, 2019.

Scope

The scope of the project is MainCoin smart contracts, which can be found on Etherscan by the link below:

https://etherscan.io/address/0x9F0F1BE08591AB7D990FAF910B38ED5D60E4D5BF

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

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level
Executive Summary

According to the assessment, Customer’s smart contracts are well-secured.

Our team performed analysis of code functionality, manual audit and automated checks with Mythril, Slither and remix IDE (see Appendix B pic 1-6). All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in Audit overview section. General overview is presented in AS-IS section and all found issues can be found in Audit overview section.

We found 1 low vulnerability in smart contract. We also outline 2 code style issues and 1 best practice recommendation.

Graph 1. The distribution of vulnerabilities.
Severity Definitions

<table>
<thead>
<tr>
<th>Risk Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical</td>
<td>Critical vulnerabilities are usually straightforward to exploit and can lead to tokens lose etc.</td>
</tr>
<tr>
<td>High</td>
<td>High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial functions</td>
</tr>
<tr>
<td>Medium</td>
<td>Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose</td>
</tr>
<tr>
<td>Low</td>
<td>Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution</td>
</tr>
<tr>
<td>Lowest / Code Style / Best Practice</td>
<td>Lowest-level vulnerabilities, code style violations and info statements can’t affect smart contract execution and can be ignored.</td>
</tr>
</tbody>
</table>

AS-IS overview

MainCoin contract overview

MainCoin is a simple ERC20 token contract that inherits BurnableToken and Ownable from OpenZeppelin repository. It uses SafeMath library for handling math operations.

MainCoin's constructor invokes _mint from StandardToken.

MainCoin describes ERC20 token with next parameters:

- symbol - MNC
- name - MainCoin
- decimals - 18
- initial - 500 ether * 10 ** 6

RewardsToken has 1 modifier:
• **isTrasferAllowed** - checks whether **unfrozen** is not **false** or **allowedAddresses** is **true** for address a or b.

**MainCoin** has 10 functions:

• **allowTransfer** is a public function – allows transfer for specified address. Has **onlyOwner** modifier.

• **disableTransfer** is a public function – disables transfer for specified address. Has **onlyOwner** modifier.

• **unfreeze** is a public function – sets **unfrozen** to **true**. Has **onlyOwner** modifier. Has **isTrasferAllowed** modifier.

• **burn** is a public function – burns specified amount of tokens from **msg.sender** address. Has **isTrasferAllowed** modifier.

• **burnFrom** is a public function – burns specified amount of tokens from specified address. Has **isTrasferAllowed** modifier.

• **transferFrom** is a public function – transfers specified amount of tokens from one specified address to another. Has **isTrasferAllowed** modifier.

• **transfer** is a public function – transfers specified amount of tokens to the specified address. Has **isTrasferAllowed** modifier.

• **approve** is a public function – approves specified amount of tokens to the specified address. Has **isTrasferAllowed** modifier.
• `increaseApproval` is a public function – increases approved amount of tokens to the specified address. Has `isTransferAllowed` modifier.

• `decreaseApproval` is a public function – decreases approved amount of tokens to the specified address. Has `isTransferAllowed` modifier.
Audit overview

Critical

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

No medium severity vulnerabilities were found.

Low

1. Compiler version is not locked. It is recommended to lock the compiler version with the latest one (see Appendix A pic 1 for evidence).

   \texttt{pragma solidity ^0.4.24; // bad: compiles w 0.4.24 and above}

   \texttt{pragma solidity 0.5.7; // good: compiles w 0.5.7 only}

Lowest / Code style / Best Practice

Lowest

No lowest severity vulnerabilities were found.

Code style

2. Visibility modifier must be first in list of modifiers.

3. Line length must be no more than 120 but it is higher in:
   
   - \texttt{MainCoin} on line 470 and 475
Best practice

4. All OpenZeppelin contracts should be updated to the latest version for gas optimizations.
**Conclusion**

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. For the contract high level description of functionality was presented in As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Overall quality of reviewed contracts is good. Security engineers found 1 low vulnerability, which couldn’t have any significant security impact.
**Disclaimers**

**Hacken Disclaimer**

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

The audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only – we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

**Technical Disclaimer**

Smart contracts are deployed and executed on blockchain platform. The platform, its programming language, and other software related to the smart contract can have own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.
Appendix A. Evidences

Pic 1. Compiler version not locked:

```solidity
pragma solidity ^0.4.24;
```
Appendix B. Automated tools reports

Pic 1. Slither automated report part 1:

For the full report, please refer to the attached file.

Pic 2. Slither automated report part 2:

For the full report, please refer to the attached file.
Pic 3. Mythril automated report:

```
user@ubuntu:~/Desktop$ myth -x main.sol
==== Integer Overflow ====
SWC ID: 101
Severity: High
Contract: StandardToken
Function name: [increaseApproval(address,uint256), increaseApproval(address,uint256), increaseApproval(address,uint256)] (ambiguous)
PC address: 4096
Estimated Gas Usage: 1044 - 1479
The binary addition can overflow.
The operands of the addition operation are not sufficiently constrained. The addition could therefore result in an integer overflow. Prevent the overflow by checking inputs or ensure sure that the overflow is caught by an assertion.

-------------
In file: main.sol:84
  _a + _b

-------------

==== Integer Overflow ====
SWC ID: 101
Severity: High
Contract: BurnableToken
Function name: [increaseApproval(address,uint256), increaseApproval(address,uint256), increaseApproval(address,uint256)] (ambiguous)
PC address: 4267
Estimated Gas Usage: 1048 - 1523
The binary addition can overflow.
The operands of the addition operation are not sufficiently constrained. The addition could therefore result in an integer overflow. Prevent the overflow by checking inputs or ensure sure that the overflow is caught by an assertion.

-------------
In file: main.sol:84
  _a + _b

-------------
```

```
Pic 4. Remix IDE Token automated report part 1:

Static Analysis raised 33 warning(s) that requires your attention. Click here to show the warning(s).

- BurnableToken
- ERC20
- MainCoin
- Ownable
- SafeMath
- StandardToken

Pic 5. Remix IDE Token automated report part 2:
Pic 6. Remix IDE Token automated report part 3:

<table>
<thead>
<tr>
<th>Issue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Successfully constant but it is not. Note: Modifications are correctly not considered by the static analysis.</td>
</tr>
<tr>
<td>File</td>
<td>Patches should be considered but is not. Note: Modifications are currently not considered by the static analysis.</td>
</tr>
<tr>
<td>File</td>
<td>Mandatory namespacing address not resolved.</td>
</tr>
<tr>
<td>File</td>
<td>Mandatory namespacing address not resolved.</td>
</tr>
<tr>
<td>File</td>
<td>Mandatory namespacing address not resolved.</td>
</tr>
<tr>
<td>File</td>
<td>Mandatory namespacing address not resolved.</td>
</tr>
<tr>
<td>File</td>
<td>Patches should be considered but is not. Note: Modifications are currently not considered by the static analysis.</td>
</tr>
<tr>
<td>File</td>
<td>Patches should be considered but is not. Note: Modifications are currently not considered by the static analysis.</td>
</tr>
<tr>
<td>File</td>
<td>Patches should be considered but is not. Note: Modifications are currently not considered by the static analysis.</td>
</tr>
<tr>
<td>File</td>
<td>Sensors have very similar names + and 2 +. Note: Modifications are currently not considered by the static analysis.</td>
</tr>
<tr>
<td>File</td>
<td>Sensors have very similar names + and 2 +. Note: Modifications are currently not considered by the static analysis.</td>
</tr>
<tr>
<td>File</td>
<td>Sensors have very similar names + and 2 +. Note: Modifications are currently not considered by the static analysis.</td>
</tr>
<tr>
<td>File</td>
<td>Sensors have very similar names + and 2 +. Note: Modifications are currently not considered by the static analysis.</td>
</tr>
</tbody>
</table>

Use assertion if you never ever want it to be false, not in any circumstance (apart from a bug in your code). Use requirement if it can be false, due to e.g. inverse logic or a failing external component.