

# Smart Contract Secondary Code Review And Security Analysis Report

**Customer:** Current (Gibraltar) Ltd

**Date:** 16.08.2018

**HACKEN**

HACKEN CYBERSECURITY ECOSYSTEM.

This document contains confidential information about IT systems and intellectual property of the customer as well as information about potential vulnerabilities and methods of their exploitation.

This confidential information shall be used only internally by the customer and shall not be disclosed to third parties.

### Document:

<b>Name</b>	Smart Contract Secondary Code Review and Security Analysis Report for Current (Gibraltar) Ltd
<b>Platform</b>	Ethereum / Solidity
<b>Link</b>	<a href="https://github.com/CurrentMediaNetwork/CRNC/tree/master/contracts">https://github.com/CurrentMediaNetwork/CRNC/tree/master/contracts</a>
<b>Date of first audit</b>	16.08.2018
<b>Version of first audit</b>	6551329ef1780d855ca9b0f6970b6d6480edd939
<b>Date version of secondary audit</b>	17.08.2018
<b>Version of secondary audit</b>	d7f508cedda739325e6ad40e5531fa68cdf43d0e

## *Table of contents*

Introduction.....	4
Scope.....	4
Executive Summary .....	5
Severity Definitions.....	6
AS-IS overview .....	6
Audit overview .....	9
Conclusion .....	10
Disclaimers .....	10
Appendix A. Evidences .....	11
Appendix B. Automated tools reports.....	12

## Introduction

Current (Gibraltar) Ltd (Customer) contacted Hacken OÜ (Consultant) to conduct 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 10.08.2018 – 16.08.2018. Secondary audit was conducted between 16.08.2018 – 17.08.2018.

## Scope

This project concerns Current (Gibraltar) Ltd smart contracts, which can be found on Github at the link below:

<https://github.com/CurrentMediaNetwork/CRNC/tree/master/contracts>

Commit version: `6551329ef1780d855ca9b0f6970b6d6480edd939`

Secondary audit was conducted for Current (Gibraltar) Ltd smart contracts within next commit:

Commit version: `d7f508cedda739325e6ad40e5531fa68cdf43d0e`

The full list of audited contracts is `CurrentToken.sol`, `Custodial.sol`, `Pausable.sol`, `PausableToken.sol`.

We have scanned this smart contract for common and company-specific vulnerabilities. The following list includes common vulnerabilities that were considered:

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

## Executive Summary

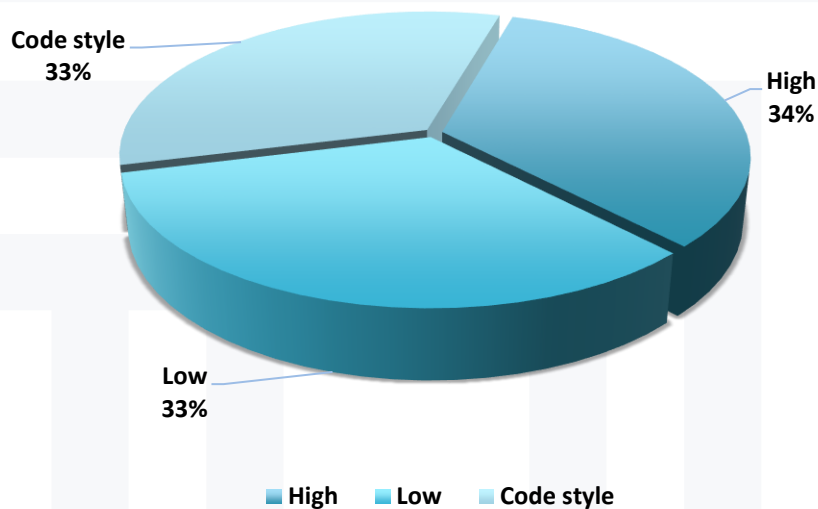
According to our assessment, the level of the smart contracts' security is high.



Our team performed an analysis of the code's functionality, manual audit, and automated checks with solc, Mythril and remix IDE (see Appendix B pic 1-8). All the issues discovered by the automated tools analysis were manually reviewed, and valid vulnerabilities are presented in the Audit overview section. The general overview is presented in the AS-IS section, and all discovered issues can be found in the Audit overview section.

We found 1 high-level vulnerability, 1 low-level vulnerability, and 1 code style issue – all of them were resolved and during secondary audit no security vulnerabilities were found.

Graph 1. The distribution of vulnerabilities.



## Severity Definitions

Risk Level	Description
<b>Critical</b>	Critical vulnerabilities are usually straightforward to exploit and can directly lead to token loss
<b>High</b>	High-level vulnerabilities are difficult to exploit; however, they have a significant impact on the execution of the smart contract (e.g. public access to crucial functions)
<b>Medium</b>	Medium-level vulnerabilities have moderate security impact; however, they can't lead to token loss
<b>Low</b>	Low-level vulnerabilities are mostly related to outdated or unused code snippets that can't have a significant impact on the execution
<b>Lowest / Code Style / Info</b>	Lowest-level vulnerabilities, code style violations, and info statements can't affect the execution of a smart contract and can be ignored.

## AS-IS overview

### CurrentToken contract overview

The [CurrentToken.sol](#) contract contains the [SafeMath](#) library, the [ERC20](#) contract, the [Pausable](#) contract, the [PausableToken](#) contract, the [Custodial](#) contract, the [StandardToken](#) contract, and the [CurrentToken](#) contract.

The [SafeMath](#) library defines Math operations with safety checks that detect errors.

The [ERC20](#) contract defines interface according to the ERC20 standard.

The [StandardToken](#) contract is based on the [ERC20](#) contract and defines parameters of functions according to the ERC20 standard: mapping [allowed](#), the function [transferFrom](#), the function [approve](#), the function [allowance](#), the function [increaseApproval](#), the function [decreaseApproval](#).

The [Custodial](#) contract implements basic authorization control of functions.

The [Pausable](#) contract is based on the [Custodial](#) contract and allows to implement a stop mechanism.

The [PausableToken](#) contract inherits [StandardToken](#) and [Pausable](#) contracts. It is a modified [StandardToken](#) with pausable transfers.

The [CurrentToken](#) contract is based on the [PausableToken](#) contract and describes custom ERC20 tokens for MobileX Labs.

The `CurrentToken.sol` contract describes a custom ERC20 token with the following parameters:

- name – Current
- symbol – CRNC
- decimals – 18

The `CurrentToken` contract constructor sets:

- communityAddress \_to \_communityAddress
- presaleAddress to \_presaleAddress
- gibraltarAddress to \_gibraltarAddress
- distributorAddress to \_distributorAddress
- totalSupply\_ to 100000000000000000000000000000000
- balances[communityAddress] to \_communityTokens
- balances[presaleAddress] to \_presaleTokens
- balances[gibraltarAddress] to \_gibraltarTokens

CurrentToken.sol has 2 functions:

- `batchTransfer` is a public function – transfers batches with a specified number of tokens to a specified array of addresses. It has the `whenNotPaused` modifier.
- `batchTransferWhenPaused` is a public function – transfers batches with a specified number of tokens to a specified array of addresses. It functions only if the `msg.sender` is whitelisted and has the `whenPaused` modifier.

## Custodial contract overview

The `Custodial.sol` contract provides the `onlyCustodian` modifier, which prevents anyone from running functions on a non-custodian account. `Custodial` is an analog of the OpenZeppelin `Ownable` contract.

## Custodial contract constructor sets:

- custodian to custodian

Custodial.sol has 1 modifier:

- `onlyCustodian` – checks whether `msg.sender` is a custodian.

Custodial.sol has 2 functions:

- `renounceCustody` is a public function – changes `custodian` to the `address(0)`. It has the `onlyCustodian` modifier.
- `transferCustody` is a public function – changes `custodian` to a specified address. It has the `onlyCustodian` modifier.



## *Overview of changes in OpenZeppelin contracts*

`onlyOwner` modifier was changed to `onlyCustodian` in `Pausable.sol`. Internal constructor from `Custodial` was added.





## Audit overview

### Critical

No critical severity vulnerabilities were found.

### High [Fixed]

1. The `CurrentToken` constructor sets addresses for `_communityTokens`, `_presaleTokens`, and `_gibraltarTokens`; after that, it redistributes tokens to these accounts. In case the same addresses are passed to the constructor, account balances will be overridden, which will cause token loss. To solve the issue, Customer needs to add several checks to the code: checks that ascertain that `_communityTokens`, `_presaleTokens`, and `_gibraltarTokens` don't have the same addresses and checks that exclude the `0x0` addresses (See Appendix A pic. 1 for evidence).

Fixed in `d7f508c`: Checks for addresses were added.

### Medium

No medium severity vulnerabilities were found.

### Low [Fixed]

2. The compiler version is not locked. Consider locking the compiler version with the latest one (See Appendix A pic. 2 for evidence).

```
pragma solidity ^0.4.24; // bad: compiles w 0.4.24 and above
pragma solidity 0.4.24; // good: compiles w 0.4.24 only
```

Fixed in `d7f508c`: Compiler version was locked.

### Lowest / Code style / Info [Fixed]

#### Code style issues

3. The visibility modifier is not the first on the list of modifiers; it should be the first of modifiers in `CurrentToken` line 46

Fixed in `d7f508c`: Visibility modifier was moved to be first in list of modifiers.

## *Conclusion*

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Detailed description of the functionality of the contract is presented in the AS-IS overview section of the report.

The audit team has found 1 high and 1 low-security issue in the course of manual and automated audits. These vulnerabilities are listed in the audit overview section. During secondary audit, which we conducted between August 16th, 2018 – August 17th, 2018, no security issues were found. All issues that were discovered during initial audit were resolved within next release.

Overall quality of reviewed contracts is high and all found issues were fixed.

## *Disclaimers*

### *Disclaimer*

The audited smart contracts have been analyzed in accordance with the best industry practices available when this report was released. The report concerns several aspects of the smart contract source code and the related cybersecurity vulnerabilities and issues - compilation, deployment, and functionality.

The audit gives no warranties regarding the security of the code. Further, it cannot be considered a sufficient assessment of the utility and safety of the code, bugfree status, or any other statements of the contract. While the conducted analysis relied on experts and advanced techniques, it is important to note that you should not rely on this report only - we recommend to proceed with several independent audits and a public bug bounty program to ensure the security of smart contracts.

### *Technical Disclaimer*

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

## Appendix A. Evidences

Pic 1. Constructor in CurrentToken:

```

37     constructor(
38         uint256 _communityTokens,
39         uint256 _presaleTokens,
40         uint256 _gibraltarTokens,
41         address _communityAddress,
42         address _presaleAddress,
43         address _gibraltarAddress,
44         address _distributorAddress,
45         address _pausableCustodianAddress
46     ) PausableToken(_pausableCustodianAddress) public {
47         communityAddress = _communityAddress;
48         presaleAddress = _presaleAddress;
49         gibraltarAddress = _gibraltarAddress;
50         distributorAddress = _distributorAddress;
51
52         totalSupply_ = 1000000000000000000000000;
53
54         uint256 _initSupply = _communityTokens.add(_presaleTokens.add(_gibraltarTokens));
55
56         require(_initSupply == totalSupply_, "Initialized token supply does not match total supply");
57
58         balances[communityAddress] = _communityTokens;
59         balances[presaleAddress] = _presaleTokens;
60         balances[gibraltarAddress] = _gibraltarTokens;
61     }

```

Pic 2. The compiler version is not locked:

```
1 pragma solidity ^0.4.24;
```

## Appendix B. Automated tools reports

Pic 1. Solc automated report:

```
max@Hacken:~/solidity/projects/MobilexLabs$ solc -o . --bin --abi --overwrite *.sol
max@Hacken:~/solidity/projects/MobilexLabs$
```

Pic 2. Mythril CurrentToken automated report:

```
max@Hacken:~/solidity/projects/MobilexLabs$ myth -x CurrentToken.sol
==== Integer Overflow ====
Type: Warning
Contract: Unknown
Function name: batchTransfer(address[],uint256[])
PC address: 1527
A possible integer overflow exists in the function `batchTransfer(address[],uint256[])`.
The addition or multiplication may result in a value higher than the maximum representable integer.
-----
In file: CurrentToken.sol:72

function batchTransfer(
    address[] _recipients,
    uint256[] _distributions
)
public
whenNotPaused
{
    require(_recipients.length == _distributions.length, "Recipient and distribution arrays do not match");

    for (uint256 i = 0; i < _recipients.length; i++) {
        transfer(_recipients[i], _distributions[i]);
    }
}

-----

==== Integer Overflow ====
Type: Warning
Contract: Unknown
Function name: transferFrom(address,address,uint256)
PC address: 9935
A possible integer overflow exists in the function `transferFrom(address,address,uint256)`.
The addition or multiplication may result in a value higher than the maximum representable integer.
-----
In file: SafeMath.sol:52

_a + _b
-----
```

Pic 3. Mythril CurrentToken automated report:

```
==== Exception state ====
Type: Informational
Contract: Unknown
Function name: transferFrom(address,address,uint256)
PC address: 9948
A reachable exception (opcode 0xfe) has been detected. This can be caused by type errors, division by zero, out-of-bounds
array access, or assert violations. This is acceptable in most situations. Note however that 'assert()' should only be u
sed to check invariants. Use 'require()' for regular input checking.
-----
In file: SafeMath.sol:53

assert(c >= _a)
-----

max@Hacken:~/solidity/projects/Mobilexlabs$
```

Pic 4. Mythril Custodial automated report:

```
max@Hacken:~/solidity/projects/Mobilexlabs$ myth -x Custodial.sol
input files do not contain any valid contracts
max@Hacken:~/solidity/projects/Mobilexlabs$
```

Pic 5. Remix IDE automated report part 1:

Static Analysis raised 26 warning(s) that requires your attention. Click here to show the warning(s).		×
CurrentToken		×
Custodial		×
ERC20		×
Pausable		×
PausableToken		×
SafeMath		×
StandardToken		×

Pic 6. Remix IDE automated report part 2:

Gas requirement of function CurrentToken.batchTransfer(address[],uint256[]) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)	✖
Gas requirement of function CurrentToken.batchTransferWhenPaused(address[],uint256[]) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)	✖
Gas requirement of function CurrentToken.decreaseApproval(address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)	✖
Gas requirement of function CurrentToken.increaseApproval(address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)	✖
Gas requirement of function CurrentToken.name() high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)	✖
Gas requirement of function CurrentToken.symbol() high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)	✖
Gas requirement of function CurrentToken.transfer(address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)	✖
Gas requirement of function CurrentToken.transferFrom(address,address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)	✖
Gas requirement of function PausableToken.decreaseApproval(address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)	✖
Gas requirement of function PausableToken.increaseApproval(address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)	✖

Pic 7. Remix IDE automated report part 3:

Gas requirement of function PausableToken.transfer(address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)	✖
Gas requirement of function PausableToken.transferFrom(address,address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)	✖
Gas requirement of function StandardToken.decreaseApproval(address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)	✖
Gas requirement of function StandardToken.increaseApproval(address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)	✖
Gas requirement of function StandardToken.transfer(address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)	✖
Gas requirement of function StandardToken.transferFrom(address,address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)	✖
PausableToken.transfer(address,uint256) : Potentially should be constant but is not. Note: Modifiers are currently not considered by this static analysis. <a href="#">more</a>	✖
PausableToken.transferFrom(address,address,uint256) : Potentially should be constant but is not. Note: Modifiers are currently not considered by this static analysis. <a href="#">more</a>	✖
PausableToken.approve(address,uint256) : Potentially should be constant but is not. Note: Modifiers are currently not considered by this static analysis. <a href="#">more</a>	✖
PausableToken.increaseApproval(address,uint256) : Potentially should be constant but is not. Note: Modifiers are currently not considered by this static analysis. <a href="#">more</a>	✖

Pic 8. Remix IDE automated report part 4:

PausableToken.decreaseApproval(address,uint256) : Potentially should be constant but is not. Note: Modifiers are currently not considered by this static analysis. <a href="#">more</a>	✕
SafeMath.mul(uint256,uint256) : Variables have very similar names _a and _b. Note: Modifiers are currently not considered by this static analysis.	✕
SafeMath.div(uint256,uint256) : Variables have very similar names _a and _b. Note: Modifiers are currently not considered by this static analysis.	✕
SafeMath.sub(uint256,uint256) : Variables have very similar names _a and _b. Note: Modifiers are currently not considered by this static analysis.	✕
SafeMath.add(uint256,uint256) : Variables have very similar names _a and _b. Note: Modifiers are currently not considered by this static analysis.	✕
Use assert(x) if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use require(x) if x can be false, due to e.g. invalid input or a failing external component. <a href="#">more</a>	✕