



**HACKEN**

# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Customer:** MGT Solar

**Date:** November 30, 2022

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

## Document

<b>Name</b>	Smart Contract Code Review and Security Analysis Report for MGT Solar
<b>Approved By</b>	Evgeniy Bezuglyi   SC Audits Department Head at Hacken OU
<b>Type</b>	ERC20 token
<b>Platform</b>	EVM
<b>Network</b>	BSC
<b>Language</b>	Solidity
<b>Methodology</b>	<a href="#">Link</a>
<b>Website</b>	<a href="https://www.megatechmgt.co.za">https://www.megatechmgt.co.za</a>
<b>Changelog</b>	21.10.2022 - Initial Review 30.11.2022 - Second Review



## Table of contents

Introduction	4
Scope	4
Severity Definitions	5
Executive Summary	6
Checked Items	7
System Overview	10
Findings	11
Disclaimers	13

## Introduction

Hacken OÜ (Consultant) was contracted by MGT Solar (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

## Scope

The scope of the project is smart contracts in the repository:

### Initial review scope

**Repository:**

[https://github.com/Megatech-MGT/MGT\\_token](https://github.com/Megatech-MGT/MGT_token)

**Commit:**

5f8960eb49868eb366f6b38fd96f1a23066a3c58

**Documentation:**

[Whitepaper](#)

Technical description in README.md

**Integration and Unit Tests:** Yes

**Contract address:**

<https://www.bscscan.com/address/0xb53180d39b4ca2ae290d589314a90e48f36af998>

**Contracts:**

File: ./contracts/Megatech.sol

SHA3: 0b5247aa7d7d2a086c2609ab7d95e62b1559d7552f2e126138c7dc857bfe4154

### Second review scope

**Repository:**

[https://github.com/Megatech-MGT/MGT\\_token](https://github.com/Megatech-MGT/MGT_token)

**Commit:**

7d12630347d28ba3a12e0c463275590f1cfec31a

**Documentation:**

[Whitepaper](#)

Technical description in README.md

**Integration and Unit Tests:** Yes

**Contract address:**

<https://www.bscscan.com/address/0xb53180d39b4ca2ae290d589314a90e48f36af998>

**Contracts:**

File: ./contracts/Megatech.sol

SHA3: 9273ace8b5382ee6bdc71828a4b97883993256cb8d3eb62be86c8e3a866b503e

## Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

## Executive Summary

The score measurement details can be found in the corresponding section of the [scoring methodology](#).

### Documentation quality

The total Documentation Quality score is **8** out of **10**.

- Functional requirements are provided.
- Setup instructions are provided.
- NatSpecs are not provided.

### Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.

### Test coverage

Code coverage of the project is **100%**.

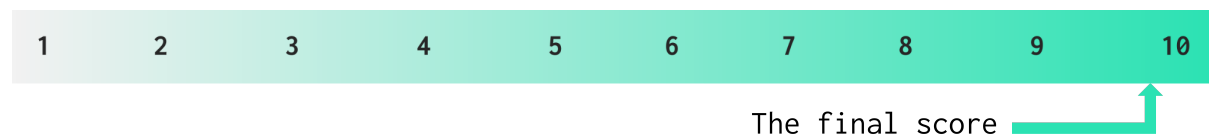
### Security score

As a result of the audit, the code contains **2** low severity issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

### Summary

According to the assessment, the Customer's smart contract has the following score: **9.8**



*Table. The distribution of issues during the audit*

Review date	Low	Medium	High	Critical
21 October 2022	4	0	2	0
28 November 2022	2	0	0	0

## Checked Items

We have audited the Customers' smart contracts for commonly known and more specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	<a href="#">SWC-100</a> <a href="#">SWC-108</a>	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	<a href="#">SWC-101</a>	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant
Outdated Compiler Version	<a href="#">SWC-102</a>	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	<a href="#">SWC-103</a>	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	<a href="#">SWC-104</a>	The return value of a message call should be checked.	Not Relevant
Access Control & Authorization	<a href="#">CWE-284</a>	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	<a href="#">SWC-106</a>	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	<a href="#">SWC-107</a>	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	<a href="#">SWC-110</a>	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	<a href="#">SWC-111</a>	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	<a href="#">SWC-112</a>	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	<a href="#">SWC-113</a> <a href="#">SWC-128</a>	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	<a href="#">SWC-114</a>	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	<a href="#">SWC-115</a>	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	<a href="#">SWC-116</a>	Block numbers should not be used for time calculations.	Not Relevant
Signature Unique Id	<a href="#">SWC-117</a> <a href="#">SWC-121</a> <a href="#">SWC-122</a> <a href="#">EIP-155</a>	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery	Not Relevant
Shadowing State Variable	<a href="#">SWC-119</a>	State variables should not be shadowed.	Passed
Weak Sources of Randomness	<a href="#">SWC-120</a>	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	<a href="#">SWC-125</a>	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	<a href="#">EEA-Leve1-2</a> <a href="#">SWC-126</a>	All external calls should be performed only to trusted addresses.	Not Relevant
Presence of unused variables	<a href="#">SWC-131</a>	The code should not contain unused variables if this is not <a href="#">justified</a> by design.	Passed
EIP standards violation	<a href="#">EIP</a>	EIP standards should not be violated.	Passed
Assets integrity	Custom	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	Custom	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	Custom	Smart contract data should be consistent all over the data flow.	Not Relevant
Flashloan Attack	Custom	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	Custom	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed



<b>Gas Limit and Loops</b>	<b>Custom</b>	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Not Relevant
<b>Style guide violation</b>	<b>Custom</b>	Style guides and best practices should be followed.	Passed
<b>Requirements Compliance</b>	<b>Custom</b>	The code should be compliant with the requirements provided by the Customer.	Passed
<b>Environment Consistency</b>	<b>Custom</b>	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
<b>Secure Oracles Usage</b>	<b>Custom</b>	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
<b>Tests Coverage</b>	<b>Custom</b>	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
<b>Stable Imports</b>	<b>Custom</b>	The code should not reference draft contracts, that may be changed in the future.	Passed

## System Overview

*MGT Solar token* is an ERC20 token with Snapshot, Permit, Votes, and Burnable options.

The token attributes:

- Name: Megatech
- Symbol: MGT
- Decimals: 18
- Total supply: 5b tokens initially and an arbitrary amount can be minted.

## Privileged roles

- Owner: The person who deploys and can use functions with onlyOwner modifier: snapshot(), mint().
- Token Owner: The person who has the token(s) and can interact with them.

## Findings

### Critical

No critical severity issues were found.

### High

#### 1. Token Supply Manipulation

The whitepaper states that the total token supply is 5 billion. However, the contract has the `mint()` function that allows minting an arbitrary amount.

Tokens should be minted only according to rules specified in the requirements.

**Path:** `./contracts/Megatech.sol : mint()`

**Recommendation:** remove the mint function.

**Status:** **Fixed** (Revised commit: 7d12630)

#### 2. Private Key Sharing

The repository contains an exposed private key.

An account may be compromised.

**Path:** `hardhat.config.js`

**Recommendation:** never store private keys in a repository. Use excluded `.env` files if needed.

**Status:** **Fixed** (Revised commit: 7d12630)

### Medium

No medium severity issues were found.

### Low

#### 1. Unused Imports

Unused imports should be removed from the contracts. This will help lower the Gas cost.

**Path:** `./contracts/Megatech.sol : SafeCast, ECDSA, EIP721, IERC20Permit, Counters, Math, Arrays, Context, IERC20, IERC20Metadata`

**Recommendation:** remove unused imports.

**Status:** **Fixed** (Revised commit: 7d12630)

#### 2. Floating Pragma

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

**Path:** ./contracts/Megatech.sol : pragma

**Recommendation:** consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

**Status:** Fixed (Revised commit: 7d12630)

### 3. Solidity Style Guide Violation

There is a [style guide in the Solidity documentation](#), which says that the arrangement of each element in the order that the SPDX-License and pragma should be above the imports, each public or external function must have NatSpec comments. However, the code from these contracts does not quite follow these recommendations.

```
report | graph (this) | graph | inheritance | parse | flatten | funcSigs | uml | draw.io
// SPDX-License-Identifier: MIT
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Snapshot.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol";

pragma solidity 0.8.7;
```

**Path:** ./contracts/Megatech.sol

**Recommendation:** follow the Solidity style guide.

**Status:** Reported

### 4. Magic Number Pattern

The code contains the number 5 billion, which is not designated as a variable. This makes it difficult to understand the code.

**Path:** ./contracts/Megatech.sol

**Recommendation:** follow the official Solidity guidelines.

**Status:** Fixed (Revised commit: 7d12630)

### 5. State Variables Default Visibility

The explicit visibility makes it easier to catch incorrect assumptions about who can access the variable.

**Path:** ./contracts/Megatech.sol : totalSupply\_

**Recommendation:** specify variables as public, internal, or private. Explicitly define visibility for all state variables.

**Status:** New

## Disclaimers

### Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with 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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

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 the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

### 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 can have vulnerabilities that can lead to hacks. Thus, Consultant cannot guarantee the explicit security of the audited smart contracts.