

**HACKEN**

# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Customer:** EWS Labs, Inc  
**Date:** April 21<sup>st</sup>, 2022



This document 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 containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed – upon a decision of the Customer.

## Document

<b>Name</b>	Smart Contract Code Review and Security Analysis Report for EWS Labs, Inc.
<b>Approved By</b>	Evgeniy Bezuglyi   SC Department Head at Hacken OU
<b>Type</b>	ERC20 token
<b>Platform</b>	EVM
<b>Language</b>	Solidity
<b>Methods</b>	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
<b>Website</b>	<a href="https://ethereumtowers.com/">https://ethereumtowers.com/</a>
<b>Timeline</b>	18.04.2022 - 21.04.2022
<b>Changelog</b>	21.04.2022 - Initial Review



## Table of contents

Introduction	4
Scope	4
Severity Definitions	5
Executive Summary	6
Checked Items	7
System Overview	10
Findings	11
Disclaimers	12

## Introduction

Hacken OÜ (Consultant) was contracted by EWS Labs, Inc (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

**Technical Documentation:** Yes

([https://drive.google.com/file/d/14N4PpF0bntnDY4Mt5qhavKpRzZ\\_06CFG/view](https://drive.google.com/file/d/14N4PpF0bntnDY4Mt5qhavKpRzZ_06CFG/view))

**JS tests:** No

**Contracts:**

<https://ropsten.etherscan.io/address/0xd252Fb4a05f16615039B2D5e52f957E8e1413120#code>

## Severity Definitions

Risk Level	Description
<b>Critical</b>	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
<b>High</b>	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
<b>Medium</b>	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
<b>Low</b>	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution

## Executive Summary

The Score measurements details can be found in the corresponding section of the [methodology](#).

### Documentation quality

The Customer provided functional requirements with tokenomics and technical requirements. The total Documentation Quality score is **10** out of **10**.

### Code quality

The total CodeQuality score is **10** out of **10**. The code follows official language style guides.

### Architecture quality

The architecture quality score is **10** out of **10**. The architecture is clear.

### Security score

As a result of the audit, security engineers found **no** 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: **10**



## Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are 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.	Passed
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 destroyed until it has funds belonging to users.	Not Relevant
Check-Effect-I interaction	<a href="#">SWC-107</a>	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Not Relevant
Uninitialized Storage Pointer	<a href="#">SWC-109</a>	Storage type should be set explicitly if the compiler version is < 0.5.0.	Not Relevant
Assert Violation	<a href="#">SWC-110</a>	Properly functioning code should never reach a failing assert statement.	Not Relevant
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 it is required.	Not Relevant

Race Conditions	<a href="#">SWC-114</a>	Race Conditions and Transactions Order Dependency should not be possible.	Not Relevant
Authorization through tx.origin	<a href="#">SWC-115</a>	tx.origin should not be used for authorization.	Passed
Block values as a proxy for time	<a href="#">SWC-116</a>	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	<a href="#">SWC-117</a> <a href="#">SWC-121</a> <a href="#">SWC-122</a>	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	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.	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.	Not Relevant
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.	Passed
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.	Passed
<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 requirements provided by the Customer.	Passed
<b>Repository Consistency</b>	<b>Custom</b>	The repository should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Not Relevant
<b>Tests Coverage</b>	<b>Custom</b>	The code should be covered with unit tests. Tests coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Not Relevant



## System Overview

EthereumWorlds is a token project with the following contract:

- EthereumWorlds – simple ERC-20 token that mints all initial supply to a distributor. Additional minting is not allowed.

It has the following attributes:

- Name: Ethereum Worlds
- Symbol: TWR
- Decimals: 18
- Total supply: 1b tokens.

### Privileged roles

- The distributor of the EthereumWorlds is the one whom the total supply is minted to when deploying.

## Findings

### ■ ■ ■ ■ Critical

No critical severity issues were found.

### ■ ■ ■ High

No high severity issues were found.

### ■ ■ Medium

No medium severity issues were found.

### ■ Low

No low severity issues were found.



## 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 audit makes no statements or warranties on the security of the code. It also cannot be considered a 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.

### 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, the audit cannot guarantee the explicit security of the audited smart contracts.