Smart Contract Audit
World Token

Versión	1.1
Auditors	David Ortega Tino Martinez
Date	03 October 2023

03 October 2023

03 October 2023

1. Dekalabs

At Dekalabs we are Software Engineers specialized in SmartContract development and audits. Amongst the team we have vast experience in dApp development and SmartContracts since 2016 on the following platforms:

- Ethereum/Quorum
- BSC
- Polygon/MATIC
- Arbitrum
- Stellar
- Algorand
- Hyperledger Fabric
- Hyperledger Besu

We have developed some top notch blockchain technology projects as climatetrade.com or stadioplus.com

We also are professors of blockchain and smart contracts in several business schools like EDEM in Valencia.

2. Legal warning

The purpose of this audit is not to assure that the Smart Contracts are error-free and that it is not possible to run an attack against them when they are online. The aim is to perform an exhaustive analisis of the source code to try to find known vulnerabilities and make the code safer in order to minimize the risk of security problems in the project.

Given that, **the information that is shown in this audit is for general analysis, code improvement and lower the risk of vulnerabilities, but it's objective is not provide legal protection to any individual or entity to the execution of the Smart Contracts.**

03 October 2023

3. Functional introduction

This document has an analysis on the security and code quality of the Smart Contracts for World Token. This token will be based on the ERC-20 standard token.

4. Audit process

In order to realize the security and code quality analysis we will follow several steps to validate each one of them in the code. Specifically the sequence will be the following.

- Check the code files integrity. So we can have a clear starting point.
- Create or validate the unit testings that will validate the functionality.
- Deploy the SmartContracts in a local network and in a test network and realize several attacks to check that there are not known vulnerabilities (SWC)
- Check code quality (DRY, ciclomatic complexity…)
- Final conclusions, suggestions..
- Release: code in a zip file (SHA1 integrity) and audit document (this document).

5. Contract integrity

```
5b3be1905338f6624430590db2f5487129c101da  IMintableERC20.sol
3cb30eb86253387cc724b08bd5f64421cf6155b2  VestingContract.sol
21f1c65f3db047659112520ac2f89be178f04825  WorldToken.sol
3f8f9d66083281998547ead9e2a599f5e3d049f8  IERC20.sol
87b62db9a86c0b9bbc58b51d0d2ae7a8b7688800  IERC20Metadata.sol
d67a42d9752d01096530f6d176bbb119b4e2133b  SafeERC20.sol
ad1343d6b9d677955bc4fab0aa009110b95f76fe  IERC20Permit.sol
169faebd8dc11ce16251c6075421d8f9fa126c58  Address.sol
691ac8cc8ecc93fa144beb50c3b0263300d15321  Ownable.sol
719844505df30bda93516e78eab1ced3bfe9ff4a  Context.sol
88ee5db718680555f1adec6b749080d6ae5aa919  ERC20.sol
bb6ee815f90a506ecb8e0ca9acd4ce91d8869f51  ERC20Burnable.sol
6a741dea32e6491a70ad425e8edf9694c7af13a8  ERC20Pausable.sol
cf3c964c64cc1a83982c1699b029c2b6931f55a7  Pausable.sol
e3c68e62977b44074fed4b60e0b0858b24baab3f  AccessControlEnumerable.sol
e3c68e62977b44074fed4b60e0b0858b24baab3f  AccessControlEnumerable.sol
22b9368b8773860feb3463c66fc58a17b45f924c  IAccessControlEnumerable.sol
d1b98c816b89db61d1c037dfe644583cccdd65be  AccessControl.sol
ff5ece3767b1d883dd71e4d905878d3af39eaa3e  Strings.sol
b3cc6713a4ecd5a40a432dd8a7382c609564ee1a  ERC165.sol
9094b1239c3bebc099412e6250d7275fcae0fdb6  EnumerableSet.sol
```

SHA1 sum is included so we can validate the integrity of them. If any change is made on the files the hash will change and this audit will be invalidated.

6. Tests

The project included the some functional tests to check the current functionality of the contracts. We have checked the validated functionality and ran the tests. After some small modifications, the code coverage of the tests is enough to check all the contract actions.

```
Compiled 1 Solidity file successfully
·------------------------|--------------------------|----------------·
|  Solc version: 0.8.16  ·  Optimizer enabled: true  ·  Runs: 800     |
·························|··························|················
|  Contract Name         ·  Size (KiB)              ·  Change (KiB)  |
·························|··························|················
|  Address               ·                   0.084  ·         0.000  |
·························|··························|················
|  console               ·                   0.084  ·         0.000  |
·························|··························|················
|  EnumerableSet         ·                   0.084  ·         0.000  |
·························|··························|················
|  ERC20                 ·                   2.250  ·         0.000  |
·························|··························|················
|  Math                  ·                   0.084  ·         0.000  |
·························|··························|················
|  MockERC20             ·                   2.481  ·         0.000  |
·························|··························|················
|  SafeERC20             ·                   0.084  ·         0.000  |
·························|··························|················
|  Strings               ·                   0.084  ·         0.000  |
·························|··························|················
|  VestingContract       ·                   4.482  ·        +0.009  |
·························|··························|················
|  WorldToken            ·                   6.604  ·         0.000  |
·------------------------|--------------------------|----------------·


 VestingContract
   constructor
     ✔ should set token address (43ms)
   createVestingSchedule
     ✔ should rever if beneficiary is zero address (58ms)
     ✔ should revert if amount is zero
     ✔ should revert if start time is in the past
     ✔ should transfer tokens to vesting contract (82ms)
     ✔ should create vesting schedule (57ms)
   vestedAmount
     ✔ should return 0 if vesting has not started (59ms)
     ✔ should return 0 if duration is 0 and vesting has not started (71ms)
     ✔ should return the total amount if duration is 0 and vesting has ended (57ms)
     ✔ should return the total amount if vesting has ended (74ms)
     ✔ should return the correct amount if vesting is in progress (764ms)
   releasableAmount
     ✔ should correctly return the releasable amount (175ms)
   release
     ✔ should rever it beneficiary has no vesting schedules
     ✔ should not send tokens if there are no tokens to release (83ms)
   getReleaseableAmount
     ✔ should return 0 if beneficiary has no vesting schedules
```

```
  15 passing (6s)
```

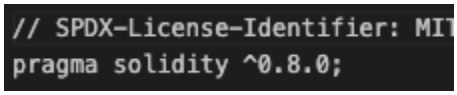
We include the test execution and the code of them to validate the execution of all tests checking all the different aspects.

Although OpenZeppelin's Ownable and AccessControl contracts are very reputable, it is a good practice to test their implementations to ensure that no third-party dependency change affects the code implementation of the contracts and that the implementation acts as intended, as the security implications of the access control roles can be immensely compromising to the token.

7. Audit

7.1. Code recommendations

- It is a good practice to lock the Solidity version for a live deployment (use 0.8.16 instead of ^0.8.0 or >=0.8.0). Contracts should be deployed with the same compiler version and flags that they have been tested with. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs. Contract may also be deployed by others and the pragma indicates the compiler version intended by the original authors.



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
```

- Remove Hardhat console debugging import from the production contracts. It can produce incompatibilities and undesired side effects.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol";
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

import "./IMintableERC20.sol";
import "hardhat/console.sol";
```

- Remove redundant variable durationUnits from VestingSchedule, as it can be managed on the deployment side to optimize the computational and storage induced gas costs. Instead, the lowest expected duration unit should be used, presumably days.

```
struct VestingSchedule {
    // beneficiary of tokens after they are released
    address beneficiary;
    // start time of the vesting period
    uint256 start;
    // duration of the vesting period in DurationUnits
    uint256 duration;
    // units of the duration
    DurationUnits durationUnits;
    // total amount of tokens to be released at the end of the vesting;
    uint256 amountTotal;
    // amount of tokens released
    uint256 released;
    // block vesting
    bool block;
}
```

- Avoid shadowing protected word block in VestingSchedule. Potentially replaceable with blocked.
- Avoid usage of require statements in favor of revert with custom error schemas for gas optimizations. (https://ethereum.stackexchange.com/questions/123381/when-should-i-use-require-vs-custom-revert-errors).

```
// perform input checks
require(_beneficiary != address(0), "VestingContract: beneficiary is the zero address");
require(_amountTotal > 0, "VestingContract: amount is 0");
require(_start >= block.timestamp, "VestingContract: start is before current time");
```

7.2. Static code analysis

In this part we are going to detail the potential vulnerabilities found in the code and the recommendations to make the code more robust.

```
Compiled with solc
Total number of contracts in source files: 2
Source lines of code (SLOC) in source files: 20
Number of low issues: 0
Number of medium issues: 0
Number of high issues: 0

ERCs: ERC20

+----------------+-------------+-------+--------------------+--------------+----------+
|      Name      | # functions |  ERCS |     ERC20 info     | Complex code | Features |
+----------------+-------------+-------+--------------------+--------------+----------+
| IMintableERC20 |      7      | ERC20 |     ∞ Minting      |      No      |          |
|                |             |       | Approve Race Cond. |              |          |
|                |             |       |                    |              |          |
+----------------+-------------+-------+--------------------+--------------+----------+
INFO:Printers:
Compiled with solc
Total number of contracts in source files: 9
Source lines of code (SLOC) in source files: 378
Number of low issues: 5
Number of medium issues: 2
```

03 October 2023

```
Number of high issues: 0

ERCs: ERC20

+-----------------+-------------+-------+--------------------+--------------+--------------------+
|       Name      | # functions | ERCS  |     ERC20 info     | Complex code |      Features      |
+-----------------+-------------+-------+--------------------+--------------+--------------------+
|  IMintableERC20 |      7      | ERC20 |     ∞ Minting      |      No      |                    |
|                 |             |       | Approve Race Cond. |              |                    |
|                 |             |       |                    |              |                    |
| VestingContract |      17     |       |                    |      No      | Tokens interaction |
|  IERC20Metadata |      9      | ERC20 |     No Minting     |      No      |                    |
|                 |             |       | Approve Race Cond. |              |                    |
|                 |             |       |                    |              |                    |
|   IERC20Permit  |      3      |       |                    |      No      |                    |
|    SafeERC20    |      7      |       |                    |      No      |      Send ETH      |
|                 |             |       |                    |              | Tokens interaction |
|     Address     |      13     |       |                    |      No      |      Send ETH      |
|                 |             |       |                    |              |    Delegatecall    |
|                 |             |       |                    |              |      Assembly      |
+-----------------+-------------+-------+--------------------+--------------+--------------------+
INFO:Printers:
Compiled with solc
Total number of contracts in source files: 17
Source lines of code (SLOC) in source files: 774
Number of low issues: 2
Number of medium issues: 10
Number of high issues: 0

ERCs: ERC165, ERC20

+---------------+-------------+--------------+--------------------+--------------+----------+
|      Name     | # functions |     ERCS     |     ERC20 info     | Complex code | Features |
+---------------+-------------+--------------+--------------------+--------------+----------+
|   WorldToken  |      74     | ERC20,ERC165 |      Pausable      |      No      |          |
|               |             |              |     ∞ Minting      |              |          |
|               |             |              | Approve Race Cond. |              |          |
|               |             |              |                    |              |          |
|    Strings    |      5      |              |                    |      No      | Assembly |
|      Math     |      14     |              |                    |     Yes      | Assembly |
| EnumerableSet |      24     |              |                    |      No      | Assembly |
+---------------+-------------+--------------+--------------------+--------------+----------+


INFO:Detectors:
VestingContract.release().totalRelease (contracts/VestingContract.sol#139) is a local variable
never initialized
VestingContract.vestedAmount(VestingContract.VestingSchedule).sliceInSeconds
(contracts/VestingContract.sol#197) is a local variable never initialized
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

INFO:Detectors:
VestingContract.release() (contracts/VestingContract.sol#133-158) has external calls inside a
loop: token.mint(schedule.beneficiary,amountToSend) (contracts/VestingContract.sol#153)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

INFO:Detectors:
Reentrancy in VestingContract.release() (contracts/VestingContract.sol#133-158):
        External calls:
        - token.mint(schedule.beneficiary,amountToSend) (contracts/VestingContract.sol#153)
        Event emitted after the call(s):
        - TokensReleased(_beneficiary,totalRelease) (contracts/VestingContract.sol#157)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

INFO:Detectors:
```

```
VestingContract.createVestingSchedule(address,uint256,uint256,VestingContract.DurationUnits,ui
nt256) (contracts/VestingContract.sol#102-128) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_start >= block.timestamp,VestingContract: start is before
current time) (contracts/VestingContract.sol#112)
VestingContract.release() (contracts/VestingContract.sol#133-158) uses timestamp for
comparisons
        Dangerous comparisons:
        - amountToSend > 0 (contracts/VestingContract.sol#147)
VestingContract.vestedAmount(VestingContract.VestingSchedule)
(contracts/VestingContract.sol#189-213) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp >= _schedule.start (contracts/VestingContract.sol#191)
        - block.timestamp < _schedule.start (contracts/VestingContract.sol#205)
        - block.timestamp >= _schedule.start + _schedule.duration * sliceInSeconds
(contracts/VestingContract.sol#207)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

INFO:Detectors:
WorldToken.constructor(string,string).name (contracts/WorldToken.sol#40) shadows:
        - ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#62-64)
(function)
        - IERC20Metadata.name()
(node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#17) (function)
WorldToken.constructor(string,string).symbol (contracts/WorldToken.sol#40) shadows:
        - ERC20.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#70-72)
(function)
        - IERC20Metadata.symbol()
(node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#22) (function)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
```

4 issues are raised, but none of them represents a risk with the current code implementation:

- Local variables not initialized: current implementation does not lead to any non-initialized variable.
- External calls in a for loop: both the loop usage and the mint calls should be safe to use in any chain within a reasonable number of release schedules per user. A reasonable number of release schedules could be understood as lower than 50.
- Timestamp comparisons: they should be avoided when dealing with short time increment validations. The risk to consider with this implementation is a ~15 second vesting advancement, which can be negligible.
- Variable shadowing in constructor: it is a good practice to use _variable naming convention with constructor arguments to avoid undesired side effects. None were found with the current implementation.

7.3. Smart Contract Weakness Classification (SWCs)

In this case we have found the following issues, depending on severity:

Low	Medium	High
2	0	0

The Low level severity issues are the following:

SWC-103: A floating pragma is set.

This issue appears on both VestingContract.sol and WorldToken.sol contracts. As has been explained in section 7.1.

8. Code quality

As said in the point 6 and 7 there are several improvements to do in the code, specifically:

- Developing unitary tests for third-party implemented access control roles and actions.
- Updating all compiler versions to the same and removing the floating pragmas (>= and ^) so the code compiles only on a fixed version and compiler problems are avoided.
- Renaming overshadowing variables.
- Using custom errors with reverts instead of require statements for gas optimizations.
- Updating the code so it uses correctly the libraries as OpenZeppelin. If there is an ERC20, for example, it has to be always based on the same library.

9. Conclusion

All issues found in this audit were low severity. The recommendation would be to address those issues and following recommendations before going live.

10. Deliverables

The final deliverables for the audit are:

- Audit document (this document).
- Hardhat project audited

11. Update v1.1

All the low severity issues were addressed and resolved in 5e7079ddb47be1da8190e2bc3670a773e3f7d1a5 commit.

The contract integrity section has been updated with the hashes of the audited contracts.