Smart Contract Audit
Chai Token

Versión	1.0
Auditor	David Ortega (Dekalabs)
Date	01 February 2022

01 February 2022

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.**

3. Functional introduction

This document has an analysis on the security and code quality of the Smart Contracts for Chai 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

```
cfe6b4916ac7bd14bb11bccbae3702b59f6ba181  contracts/MToken/MErc20.sol
1c25a2720e927c8702ee5bd8fb0f4655adeb9c32  contracts/MToken/MErc20Delegate.sol
879aefde1a76161ae9448939f34bda73cb8136f1  contracts/MToken/MErc20Delegator.sol
899a8ec0bbd2b2251bb58173b5c0abe18b9b1ce8  contracts/MToken/MErc20Immutable.sol
eb14125a08870bca495754bafeeaa9297e552de6  contracts/MToken/MEther.sol
075b382ed06fb04688f9ffdbca420e70683bd488  contracts/MToken/MMojitoDollar.sol
a2e6f14d1f22c1955a59f5cf68802b9a37bade17  contracts/MToken/MToken.sol
50ed069831aeef861ce2a71ab03baa4332a13cf5  contracts/common/CarefulMath.sol
d9e38ccb06e85d65b740a3e872a3a94401998297  contracts/common/ErrorReporter.sol
000caab38ea92867336ce243666c56d46b84ef0c  contracts/common/Exponential.sol
32db41ed8c75949374286bddfcbc37f25a838f32  contracts/common/ExponentialNoError.sol
ddf2848ff1adbe32af7e0b91e17769360194edf6  contracts/common/SafeMath.sol
e96f37bfe749a73bbffb4ba4651ed6fe28fa0da7  contracts/comptroller/Comptroller.sol
68db9c387fd273e87f57690b80ab2640d08ec685  contracts/comptroller/ComptrollerStorage.sol
806b4a24997a7f4badfd07f245eaebea4e63ee30  contracts/comptroller/DelegateComptroller.sol
50609344cf9610e9911248273cdaedb572f54052  contracts/comptroller/DelegateComptrollerAdminStorage.sol
d3babaf05ecc9efb99d8c26d06308a293b2d3a72  contracts/funds/ComptrollerFund.sol
a5032ad58015babbe3861e7fc8678094c3bc119a  contracts/funds/StakingFund.sol
b5755fc7f09c70a0047d54a3ddb2dad36f7f7e7d  contracts/interest/InterestRateModel.sol
eb975203bce97305daa6b68359327b29f73c6dd4  contracts/interest/JumpRateInterestModel.sol
77f8de30eec153d51f1c4bdb0c016120a922c608  contracts/interest/StableCoinInterestRateModel.sol
ba838dbbef0a5d8780d1010831677ba00176d4c6  contracts/interfaces/ComptrollerFundInterface.sol
7092140181ab83a2da3cb4bbcff54e9fa2f3212b  contracts/interfaces/ComptrollerInterface.sol
1557865715c88a7cb11f921a3a2dbc7b5c9dba71  contracts/interfaces/EIP20Interface.sol
8ed791235ad9c37d79a3211abad9c1bfb56afdb7  contracts/interfaces/EIP20NonStandardInterface.sol
e8011fffaba7baa9bb1dd78bff8d5353c5aa5458  contracts/interfaces/IFund.sol
5452aff697f6dcf4cfbb3f39c2d62475a20b1838  contracts/interfaces/MTokenInterfaces.sol
b8dc5e8433ec5039056bd1e2c4abdc4bbec302aa  contracts/interfaces/PriceOracle.sol
85420ff84ed6f160a8e90596c8baa1187c129f0e  contracts/oracle/IERC2362.sol
4a8e2f3f968c8a9f714ac2f5b3fdfbf551d7713b  contracts/oracle/WitnetOracle.sol
9b1de225f02d4d472d435dd940df3df398523382  contracts/staking/MojitoChef.sol
621ceda850b0e7e24eb84bcc245bcc0f6ab28a5b  contracts/tokens/MojitoDollar.sol
3db48d1c4c32f930a5effb2ced5042411c1ab981  contracts/tokens/MojitoToken.sol
fb0db8ba646f7604734d617150228b8c3d79b26a  contracts/utils/MEtherRepayDelegate.sol
5803110864423256df260369aa8b4995c4d01925  contracts/utils/RewardEstimator.sol
517be1232c84629d07c464e065a42c28444cc936  contracts/utils/Timelock.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. Audit

The project doesn't include any automatic tests so we have checked the code visually. The first impressions is that the code has been copied and pasted from different sources and the biggest problem is the mixing of "pragma" marks with several compilers. One project should have one and only one compiler target and the pragma mark should be a specific version of this compiler.

In this case we have four different versions specified:

```
pragma solidity >=0.5.16;
pragma solidity ^0.5.16;
pragma solidity >=0.8.0;
pragma solidity ^0.5.8;
```

This versioning problems may affect to the stability and interoperability of the contracts. In concrete, in a brief review of the code we can see that there are two different MojitoToken and also two MojitoDollar (one with 0.5 and one with 0.8). It looks like there was a previous version of contracts working with 0.5 and at some point the project added different classes (MojitoChef, ComptrollerFund, WitnetOracle, etc..) that used OpenZeppelin libraries and because of that needed 0.8 compiler version.

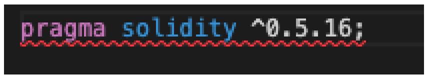
The recommendation would be moving all compiler versions to 0.8 and updating the code to match the new version. If this is not possible it would be better dividing the project in two different projects, one for 0.8 version and another with the code for 0.5. All 0.5 files should have the same pragma mark, ideally without ">=" or "^" tags, just specific versions so we don't have problems with deployments in the future.

Apart from that we can see that there is a mixed usage of the libraries. For example, in order to create ERC20 tokens in some places it is used OpenZeppelin's ERC20 interface (MojitoDollar.sol), in others, the interface has been developed directly into the contract (MojitoToken.sol) and in others the contract is using a specific interface (MToken.sol or MMojitoDollar.sol – And this two use different interfaces –).

It looks like the code has been created by mixing parts from different sources and not taking care of interrelationships or compiler versions. The first recommendation would be organizing the code, compiler versions and removing all unnecessary classes.

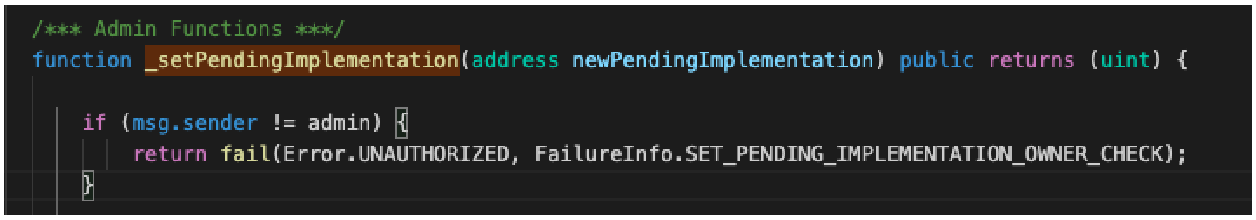
6.1. Code recommendations

- It is a good practice to lock de solidity version for a live deployment (use 0.5.16 instead of ^0.5.16 or >=0.5.16). Contracts should be deployed with the same compiler version and flags that they have been tested the most 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.



```
pragma solidity ^0.5.16;
```

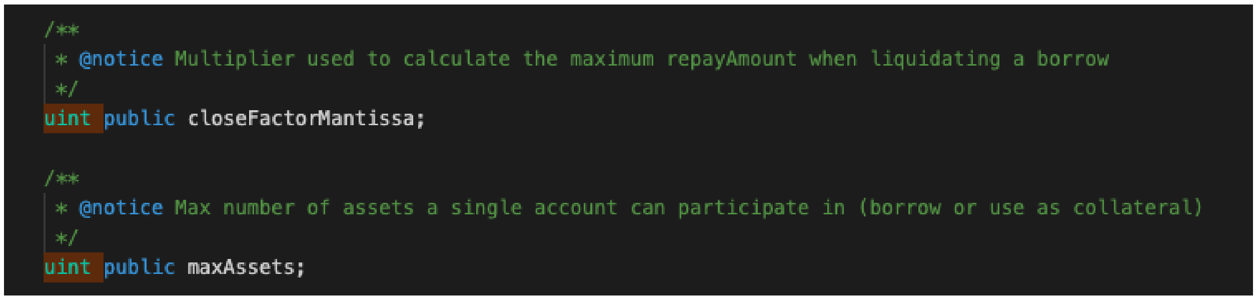
- Functions wich are declared as public and not called internally in the contract can be declared as external for two reasons: 1) Gas usage is lower in an external function compared to public and 2) increases code readability.



```
/*** Admin Functions ***/
function _setPendingImplementation(address newPendingImplementation) public returns (uint) {

    if (msg.sender != admin) {
        return fail(Error.UNAUTHORIZED, FailureInfo.SET_PENDING_IMPLEMENTATION_OWNER_CHECK);
    }
```

- Throughout the code base, some variables are declared as uint. To favor explicitness, consider changing all instances of uint to uint256.



```
/**
 * @notice Multiplier used to calculate the maximum repayAmount when liquidating a borrow
 */
uint public closeFactorMantissa;

/**
 * @notice Max number of assets a single account can participate in (borrow or use as collateral)
 */
uint public maxAssets;
```

- Trust in smart contract can be better established if their source code is available. Since making source code available always touches on legal problems with regards to copyright, it is recommended to use SPDX license identifiers. Every source should start with a comment indicating its license:

  //SPDX-License-Identifier: MIT

6.2. Slither tool

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

Number of lines: 7995 (+ 1109 in dependencies, + 0 in tests)

01 February 2022

Number of assembly lines: 53
Number of contracts: 54 (+ 8 in dependencies, + 0 tests)

Number of optimization issues: 63
Number of informational issues: 285
Number of low issues: 70
Number of medium issues: 62
Number of high issues: 7

As this code has different compiler versions and some of them are old versions the audit finds a lot of different issues and optimizations. In order to improve readability of this document the output will be attached in a different document. We are only adding here the high risk issues.

```
MEtherRepayDelegate.repayBehalfExplicit(address,MEther) (contracts/utils/MEtherRepayDelegate.sol#29-41) sends eth to arbitrary user
	Dangerous calls:
	- mEther_.repayBorrowBehalf.value(borrows)(borrower) (contracts/utils/MEtherRepayDelegate.sol#36)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations

Comptroller (contracts/comptroller/Comptroller.sol#15-1746) contract sets array length with a user-controlled value:
	- accountAssets[borrower].push(mToken) (contracts/comptroller/Comptroller.sol#194)
Comptroller (contracts/comptroller/Comptroller.sol#15-1746) contract sets array length with a user-controlled value:
	- allMarkets.push(MToken(mToken)) (contracts/comptroller/Comptroller.sol#1259)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#array-length-assignment

MErc20Delegator.delegateTo(address,bytes) (contracts/MToken/MErc20Delegator.sol#420-428) uses delegatecall to a input-controlled function id
	- (success,returnData) = callee.delegatecall(data) (contracts/MToken/MErc20Delegator.sol#421)
MErc20Delegator.fallback() (contracts/MToken/MErc20Delegator.sol#461-475) uses delegatecall to a input-controlled function id
	- (success) = implementation.delegatecall(msg.data) (contracts/MToken/MErc20Delegator.sol#465)
DelegateComptroller.fallback() (contracts/comptroller/DelegateComptroller.sol#135-147) uses delegatecall to a input-controlled function id
	- (success) = comptrollerImplementation.delegatecall(msg.data) (contracts/comptroller/DelegateComptroller.sol#137)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#controlled-delegatecall

DelegateComptrollerAdminStorage.comptrollerImplementation (contracts/comptroller/DelegateComptrollerAdminStorage.sol#20) is never initialized. It is used in:
	- Comptroller.adminOrInitializing() (contracts/comptroller/Comptroller.sol#1407-1409)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables
```

6.3. Smart Contract Weakness Classification (SWCs)

For the issues classification we are going to divide the code in two different sub-projects depending on the compiler version.

Compiler 0.5.X

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

Low	Medium	High
7	0	0

The Low level severity issues are the same:

SWC-103: Floating pragma is set.

This issue is something that we talked about in the previous sections. The compiler versions should be fixed.

Compiler 0.8.X

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

Low	Medium	High
27	0	0

The Low level severity issues are the following:

SWC-103: Floating pragma is set.

This issue is something that we talked about in the previous sections. The compiler versions should be fixed.

SWC-120: Potential use of block.number as a source of randomness.

The code, mainly in MojitoChef.sol, is using block.number. The usage is mainly as calculator of the time (difference of blocks). In getMultiplier seems to be doing the same, but it's important to notice that block.number is a value that can be known beforehand from attackers, so it should not be used as a source of randomness.

SWC-123: Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments). This problem was found in MojitoChef.sol, line 132.

```
uint256 lpSupply = pool.lpToken.balanceOf(address(this));
```

The potential problem may be that the implementation of this lpToken is not correct (not probable, as in this case is OpenZeppelin's) or that pool or lpToken are not being found and the method is failing.

Check the problem here: https://swcregistry.io/docs/SWC-123

SWC-107: Reentrancy

In the deposit and emergencyWithdraw methods there are several reads and writes of different values. This two methods are calling external functions after reading and writing persistent states after doing an external call. Even though it doesn't look dangerous it is important to review the possibility of reentrancy.

Check the problem here: https://swcregistry.io/docs/SWC-107

7. Code quality

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

- Updating all compiler versions to the same and removing the floating pragmas (>= and ^) so the code compiles only on one version and avoid compiler problems.
- 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.

8. Conclusion

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

9. Deliverables

The final deliverables for the audit are:

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