MOCHI - MOMA token

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **April 10 - April 15, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	04/12/2021	Gabi Urrutia
0.2	Document Edits	04/14/2021	Gabi Urrutia
1.0	Final Version	04/15/2021	Gabi Urrutia
1.1	Remediation Plan	04/16/2021	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

MOCHI engaged Halborn to conduct a security assessment on their Smart contract beginning on April 10th, 2021 and ending April 15th, 2021. The security assessment was scoped to the smart contract provided in the Github repository MOCHI Smart Contract and an audit of the security risk and implications regarding the changes introduced by the development team at MOCHI prior to its production release shortly following the assessments deadline. The security assessment was scoped to the smart contracts MOMA.sol.

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverables set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure smart-contract development.

1.2 AUDIT SUMMARY

The team at Halborn was provided a week for the engagement and assigned a full time security engineer to audit the security of the smart contract. The security engineers are blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit to achieve the following:

- Ensure that smart contract functions work as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified few security risks, and recommends performing further testing to validate extended safety and correctness in context to the whole set of contracts. External threats, such as economic attacks, oracle attacks, and inter-contract functions and calls should be validated for expected logic and state.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process,and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Smart Contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions(solgraph)
- Manual Assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions.(Slither)
- Testnet deployment (Truffle, Ganache)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident, and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.

4 - High probability of an incident occurring.

3 - Potential of a security incident in the long term.

2 - Low probability of an incident occurring.

1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.

4 - May cause a significant level of impact or loss.

3 - May cause a partial impact or loss to many.

2 - May cause temporary impact or loss.

1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL

9 - 8 - HIGH

7 - 6 - MEDIUM

5 - 4 - LOW

3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

IN-SCOPE:

The security assessment was scoped to the smart contracts:

- MOMA.sol

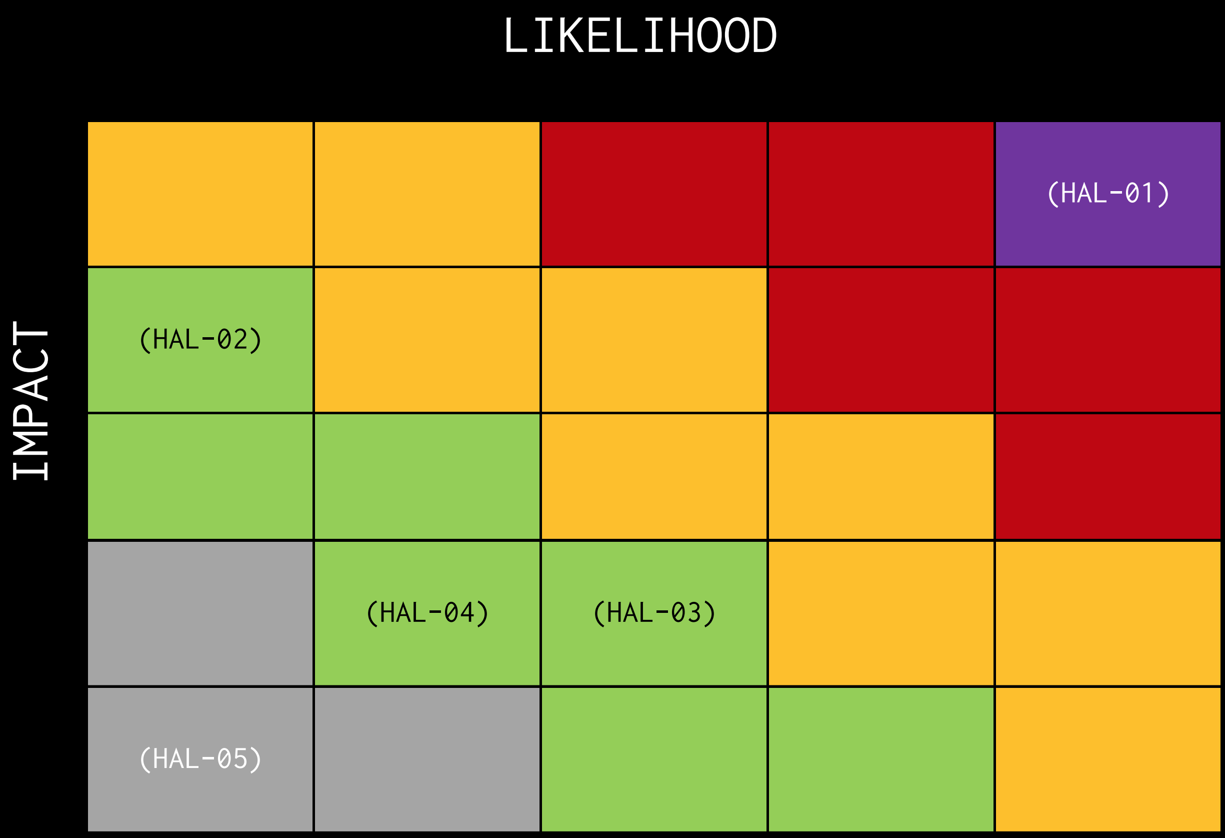
Commit ID: 6be1a4119e14b9c6223629fd22f806ee170d607e

OUT-OF-SCOPE:

Other smart contracts in the repository, external libraries and economics attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	0	0	3	1



EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - IMPROPER KEY MANAGEMENT POLICY	Critical	SOLVED: 04/16/2021
HAL02 - BLOCK TIMESTAMP ALIAS USAGE	Low	SOLVED: 04/16/2021
HAL03 - IGNORE RETURN VALUES	Low	SOLVED: 04/16/2021
HAL04 - EXPERIMENTAL FEATURES ENABLED	Low	RISK ACCEPTED: 04/16/2021
HAL05 - POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	SOLVED: 04/16/2021
MANUAL TESTING	-	-
STATIC ANALYSIS	-	-
AUTOMATED SECURITY SCAN RESULTS	-	-

FINDINGS & TECH DETAILS

3.1 (HAL-01) IMPROPER KEY MANAGEMENT POLICY - CRITICAL

Description:

A fundamental principle of blockchain is decentralization which should be applied as widely as possible in all areas, including key management. Using a single private key to manage a smart contract and perform privileged actions such as deploying or upgrading the contract is risky. If the private key is compromised, it could have devastating consequences. For example, on March 5, 2021, the PAID Network smart contract was successfully attacked despite the smart-contract being previously audited. Approximately $100 million of PAID tokens were extracted by the attacker. In that case, the private key was compromised and the attacker upgraded and replaced the original smart contract with a malicious version that allowed tokens to be burned and minted. Had best practices been implemented in the key management policy, the attacker could not have upgraded the contract using a single private key. Requiring multiple signatures in the key-management policy prevents a single user from performing any critical actions.

Reference: https://halborn.com/explained-the-paid-network-hack-march-2021/

Risk Level:

Likelihood - 5
Impact - 5

Recommendations:

Require multiple signatures in the key-management policy to avoid a private-key compromise resulting in loss of control over the smart contract.

Remediation Plan:

Solved: MOCHI team will use a multi-signature wallet for the deployment to the mainnet.

3.2 (HAL-02) BLOCK TIMESTAMP ALIAS USAGE - LOW

Description:

During a manual review, we noticed the use of block.timestamp. The contract developers should be aware that this does not mean current time. Miners can influence the value of block.timestamp to perform Maximal Extractable Value (MEV) attacks. The use of now creates a risk that time manipulation can be performed to manipulate price oracles. Miners can modify the timestamp by up to 900 seconds.

Reference: Avoid using “now”

Code Location:

MOMA.sol Line #49 #67 #68 #77 #104 #133 #135

```
48    constructor() public ERC20PresetMinterPauser("MOchi MArket", "MOMA") {
49        _blacklistEffectiveEndtime = block.timestamp + BLACKLIST_EFFECTIVE_DURATION;
50        _mint(_msgSender(), INITIAL_SUPPLY);
51    }
```

```
66    function addToBlacklist(address user) external onlyAdmin {
67        require(block.timestamp < _blacklistEffectiveEndtime, "MOMA: Force lock time ended");
68        _blacklist[user] = BlacklistInfo(true, block.timestamp, balanceOf(user));
69    }
```

```
75    function _getUnlockedBalance(address user) internal view returns (uint256 unlockedBalance) {
76        BlacklistInfo memory info = _blacklist[user];
77        uint256 daysPassed = block.timestamp.sub(info.lockedFrom).div(1 days);
78
```

```
            VestingInfo(
                true,
                amount,
                block.timestamp,
                0,
                fullLockedDays,
                releaseTotalRounds,
                daysPerRound
            );
        vestingList[beneficiary] = info;
    }
```

```
        if (!vestingList[user].isActive) return 0;
        VestingInfo memory info = vestingList[user];
        uint256 releaseTime = info.startTime.add(info.fullLockedDays.mul(1 days));
        if (block.timestamp < releaseTime) return 0;
        uint256 roundsPassed =
            (block.timestamp.sub(releaseTime)).div(1 days).div(info.daysPerRound);
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

Use block.number instead of block.timestamp or now to reduce the risk of MEV attacks. Check if the timescale of the project occurs across years, days and months rather than seconds. If possible, it is recommended to use Oracles.

Remediation Plan:

Solved: MOCHI team assumes that the use of block.timestamp is safe because their timescales are higher than 900 seconds.

3.3 (HAL-03) IGNORE RETURN VALUES - LOW

Description:

The return value of an external call is not stored in a local or state variable. In the contract MOMA.sol, the return value in withdrawERC20 is being **ignored**.

Code Location:

MOMA.sol Lines #179-183

```
179     function withdrawERC20(address token, uint256 amount) public onlyAdmin {
180         require(amount > 0, "MOMA: Amount must be greater than 0");
181         require(IERC20(token).balanceOf(address(this)) >= amount, "MOMA: ERC20 not enough balance");
182         IERC20(token).transfer( msgSender(), amount);
183     }
184
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

Add a return value check to avoid an unexpected crash of the contract. A return value check will help handle exceptions more thoroughly.

Remediation plan:

MOCHI team solved the issue in their last commit 9435238d3d80e892e7ec58cf035311ada98478a8

3.4 (HAL-04) EXPERIMENTAL FEATURES ENABLED - LOW

Description:

ABIEncoderV2 is enabled and the use of experimental features could be dangerous on live deployments. The experimental ABI encoder does not handle non-integer values shorter than 32 bytes properly. This applies to bytesNN types, bool, enum and other types when they are part of an array or a struct and encoded directly from storage. This means these storage references have to be used directly inside abi.encode(...) as arguments in external function calls or in event data without prior assignment to a local variable. The types **bytesNN** and **bool** will result in corrupted data while enum might lead to an invalid revert.

Reference: Solidity Optimizer and ABIEncoderV2 Bug

Code Location:

MOMA.sol Line #3

```
1 // SPDX-License-Identifier:GPL-3.0
2 pragma solidity 0.6.12;
3 pragma experimental ABIEncoderV2;
4
```

Reference: ConsenSys Diligence - Lock pragmas

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

When possible, do not use experimental features in the final live deployment. Validate and check that all the conditions above are true for integers and arrays (i.e. all using uint256).

Remediation Plan:

Risk Accepted: MOCHI team accepts the risk of using experimental features because they want to use the latest stable and tested version of pragma (0.6.12).

3.5 (HAL-05) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In public functions, array arguments are immediately copied to memory, while external functions can read directly from calldata. Reading call-data is cheaper than memory allocation. Public functions need to write the arguments to memory because public functions may be called internally. Internal calls are passed internally by pointers to memory. Thus, the function expects its arguments being located in memory when the compiler generates the code for an internal function.

Code Location:

Moma.sol Line #53 #179

```
53    function mint(address to, uint256 amount) public virtual override onlyMinter {
54        require(totalSupply().add(amount) <= MAX_SUPPLY, "MOMA: Max supply exceeded");
55        _mint(to, amount);
56    }
```

```
179   function withdrawERC20(address token, uint256 amount) public onlyAdmin {
180       require(amount > 0, "MOMA: Amount must be greater than 0");
181       require(IERC20(token).balanceOf(address(this)) >= amount, "MOMA: ERC20 not enough balance");
182       IERC20(token).transfer(_msgSender(), amount);
183   }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider declaring external variables instead of public variables. A best practice is to use external if expecting a function to only be called externally and public if called internally. Public functions are always accessible, but external functions are only available to external callers.

Remmediation plan:

MOCHI team solved the issue in their last commit 9435238 d3d80e892e7ec58cf035311ada98478a8

3.6 MANUAL TESTING

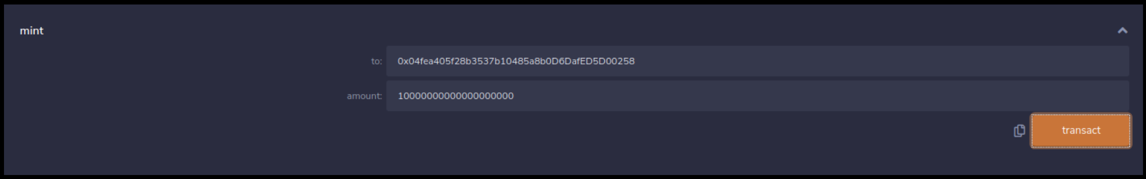
Description:

Custom tests are useful for developers to check if functions and permissions work correctly. Furthermore, they are also useful for security auditors to perform security tests behaving like a malicious user. Then, auditors manually manipulated inputs to check the security in the smart contracts.

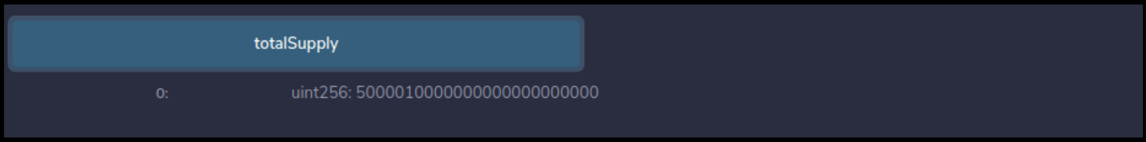
Tests:

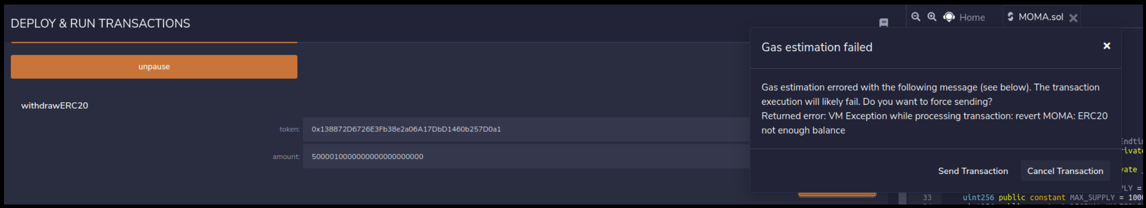
Testing if balance is checking before any transaction.

- Account[0] mint MOMA tokens for Account[1]

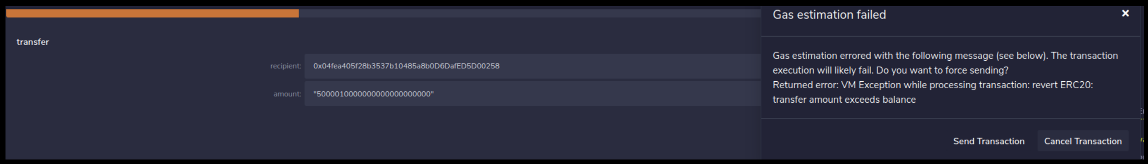


- Trying to withdraw all tokens from Account[0]

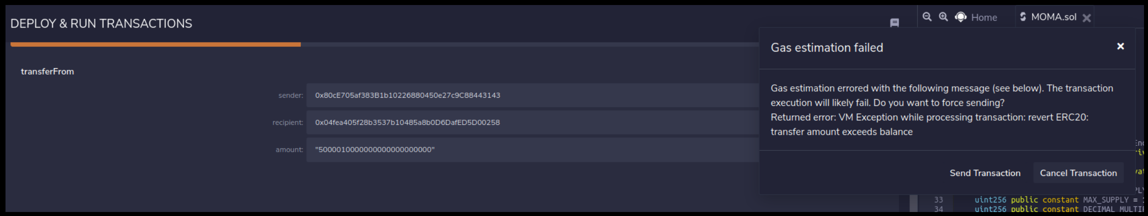




- Account[0] tries to transfer all tokens from Account[0] to Account[1].



- Account[0] tries to transfer all tokens from Account[0] to Account[1] using transferFrom function.

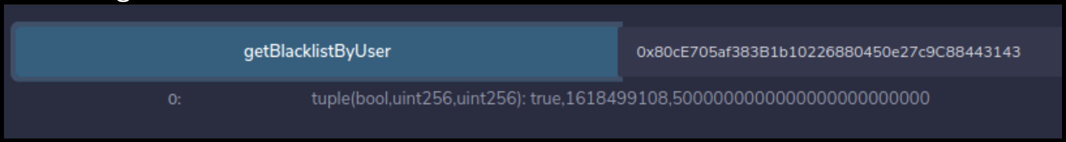


Results All transaction failed. Thus, the balance is correctly checked before making transactions

Blacklisting testing

- Owner (Account[0]) blacklists itself

- Checking if the Owner was included in the blacklist



- Trying to perform the normal actions that owner can usually does.

Results Owner can perform any action even if it blacklisted itself.

- Owner (Account[0]) blacklists Account[1]

- Owner can mint tokens in Account[1]
- Trying to perform the normal actions that Account[1] can usually does.

Results Owner can mint tokens in a blacklisted account and Account[1] can make transfer being blacklisted.

MOCHI Feedback:

The MOCHI team has clarified that the blacklist feature does not prevent an account from interacting with the contract. It is used for locking a portion of the account’s tokens, unlocking them over 50 days. This feature is to prevent trading BOTs from dumping a considerable amount of tokens right at listing.

3.7 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

```
INFO:Detectors:
MOMA._getUnlockedBalance(address) (contracts/MOMA.sol#75-85) performs a multiplication on the result of a division:
        -daysPassed = block.timestamp.sub(info.lockedFrom).div(86400) (contracts/MOMA.sol#77)
        -unlockedBalance = daysPassed.mul(info.initLockedBalance).div(BLACKLIST_LOCK_DAYS) (contracts/MOMA.sol#80)
MOMA._getVestingClaimableAmount(address) (contracts/MOMA.sol#125-147) performs a multiplication on the result of a division:
        -roundsPassed = (block.timestamp.sub(releaseTime)).div(86400).div(info.daysPerRound) (contracts/MOMA.sol#134-135)
        -releasedAmount = info.amount.mul(roundsPassed).div(info.releaseTotalRounds) (contracts/MOMA.sol#141)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
MOMA.withdrawERC20(address,uint256) (contracts/MOMA.sol#179-183) ignores return value by IERC20(token).transfer(_msgSender(),amount) (contracts/MOMA.sol#182)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
ERC20PresetMinterPauser.constructor(string,string).name (node_modules/@openzeppelin/contracts/presets/ERC20PresetMinterPauser.sol#35) shadows:
        - ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#64-66) (function)
ERC20PresetMinterPauser.constructor(string,string).symbol (node_modules/@openzeppelin/contracts/presets/ERC20PresetMinterPauser.sol#35) shadows:
        - ERC20.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#72-74) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
MOMA.addToBlacklist(address) (contracts/MOMA.sol#66-69) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp < _blacklistEffectiveEndtime,MOMA: Force lock time ended) (contracts/MOMA.sol#67)
MOMA._getUnlockedBalance(address) (contracts/MOMA.sol#75-85) uses timestamp for comparisons
        Dangerous comparisons:
        - info.locked && daysPassed < BLACKLIST_LOCK_DAYS (contracts/MOMA.sol#79)
MOMA.addVestingToken(address,uint256,uint256,uint256,uint256) (contracts/MOMA.sol#91-111) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(! _vestingList[beneficiary].isActive,MOMA: Invalid vesting) (contracts/MOMA.sol#99)
MOMA.revokeVestingToken(address) (contracts/MOMA.sol#113-119) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_vestingList[user].isActive,MOMA: Invalid beneficiary) (contracts/MOMA.sol#114)
MOMA._getVestingClaimableAmount(address) (contracts/MOMA.sol#125-147) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp < releaseTime (contracts/MOMA.sol#133)
        - roundsPassed >= info.releaseTotalRounds (contracts/MOMA.sol#138)
        - releasedAmount > info.claimedAmount (contracts/MOMA.sol#144)
MOMA.claimVestingToken() (contracts/MOMA.sol#153-162) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_vestingList[_msgSender()].isActive,MOMA: Not in vesting list) (contracts/MOMA.sol#154)
MOMA._beforeTokenTransfer(address,address,uint256) (contracts/MOMA.sol#164-177) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(balanceOf(from).sub(amount) >= lockedBalance,MOMA BLACKLIST: Cannot transfer locked balance) (contracts/MOMA.sol#172-175)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

Divide before multiply is a false positive because math operations were well implemented in the contract.

In addition, the issue regarding block.timestamp has been already raised in the report.

3.8 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruit on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the testers machine and sent the compiled results to the analyzers to locate any vulnerabilities. In addition, security detections are only in scope.

Results:

MOMA.sol

Report for contracts/MOMA.sol
https://dashboard.mythx.io/#/console/analyses/33933ca8-07f5-409c-8495-e77b13462919

Line	SWC Title	Severity	Short Description
49	(SWC-101) Integer Overflow and Underflow	High	The arithmetic operator can overflow.
67	(SWC-116) Timestamp Dependence	Low	A control flow decision is made based on The block.timestamp environment variable.

The first issue is found in `_blacklistEffectiveEndtime = block.timestamp + BLACKLIST_EFFECTIVE_DURATION;`. The value of `BLACKLIST_EFFECTIVE_DURATION` is defined by constructor so, the possible integer overflow/underflow can be considered as a false positive.
Furthermore, the issue regarding `block.timestamp` has been already raised in the report.

THANK YOU FOR CHOOSING

HALBORN