CERTIK

Security Assessment

HTMoon

May 13th, 2021

Summary

This report has been prepared for HTMoon smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	HTMoon
Description	HTMoon is the first community token issued based on the HTMoon Protocol. The HTMoon Protocol was developed by an anonymous team. The background of the HTMoon Protocol development is that the HTMoon team believes that the value of HT Token is seriously underestimated!
Platform	Heco
Language	Solidity
Codebase	https://github.com/htmoon/htmoon-contracts
Commits	f4a442f83641fdaa1f65bec0512c522b90590db5

Audit Summary

Delivery Date	May 13, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Total Issues	13
● Critical	0
● Major	1
● Medium	1
● Minor	4
● Informational	7
● Discussion	0

Audit Scope

ID	file	SHA256 Checksum
HTM	HTMoon.sol	5ea49cdf7ec1572830293bc2531ca67bc3b1aa1b98f9ea7680ef73e666403c52

Understandings

Overview

The HTMoon Protocol is a decentralized finance (DeFi) token deployed on the HecoInfo blockchain (Heco). HTMoon employs two novel features in its protocol; static rewards for each user as well as an LP acquisition mechanism. The static reward (also known as reflection) and LP acquisition mechanisms function as follows:

Each HTMoon transaction is taxed two 5% fees totaling 10% of the transaction amount. The first fee is redistributed to all existing holders using a form of rebasing mechanism whilst the other 5% is accumulated internally until a sufficient amount of capital has been amassed to perform an LP acquisition. When this number is reached, the total tokens accumulated are split with half being converted to HT and the total being supplied to the Uniswap contract as liquidity.

LP Acquisition

The LP acquisition mechanism can be indirectly triggered by any normal transaction of the token as all transfers evaluate the set of conditions that trigger the mechanism. The main conditions of the mechanism are whether the sender is different than the LP pair and whether the accumulation threshold has been breached. Should these conditions be satisfied, the `swapAndLiquify` function is invoked with the current contract's HTMoon balance.

The `swapAndLiquify` function splits the contract's balance into two halves properly accounting for any truncation that may occur. The first half is swapped to HT via the Uniswap Router using the HTMoon-HT pair and thus temporarily driving the price of the HTMoon token down. Afterward, the resulting HT balance along with the remaining HTMoon balance are supplied to the HTMoon-HT liquidity pool as liquidity via the Router. The recipient of the LP units is defined as the current owner of the HTMoon contract, a characteristic outlined in more depth within finding SSL-01.

Static Reward (Reflection)

Balances in the HTMoon token system are calculated in one of two ways. The first method, which most users should be familiar with, is a traditional fixed number of units being associated with a user's address. The second method, which is of interest to static rewards, represents a user's balance as a proportion of the total supply of the token. This method works similarly to how dynamic rebasing mechanisms work such as that of Ampleforth.

Whenever a taxed transaction occurs, the 5% meant to be re-distributed to token holders is deducted from the total "proportion" supply resulting in a user's percentage of total supply is increased. Within the system, not all users are integrated into this system and as such, the 5% fee is rewarded to a subset of the total users of the HTMoon token. The `owner` of the contract is able to introduce and exclude users from the dynamic balance system at will.

Privileged Functions

The contract contains the following privileged functions that are restricted by the `onlyOwner` modifier. They are used to modify the contract configurations and address attributes. We grouped these functions below:

Account management functions for inclusion and exclusion in the fee and reward system:

- excludeFromReward(address account)
- includeInReward(address account)
- excludeFromFee(address account)
- includeInFee(address account)

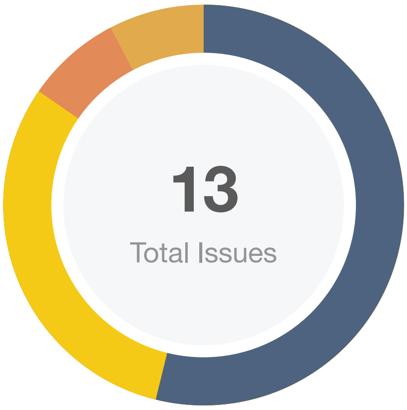
Modification of liquidation, tax and max transaction percents of the system:

- function setTaxFeePercent(uint256 taxFee)
- function setLiquidityFeePercent(uint256 liquidityFee)
- function setMaxTxPercent(uint256 maxTxPercent)

Toggle feature of the LP acquisition mechanism:

- function setSwapAndLiquifyEnabled(bool _enabled)

Findings



🟥	**Critical**	**0** (0.00%)
🟧	**Major**	**1** (7.69%)
🟧	**Medium**	**1** (7.69%)
🟨	**Minor**	**4** (30.77%)
🟦	**Informational**	**7** (53.85%)
🟩	**Discussion**	**0** (0.00%)

ID	Title	Category	Severity	Status
HTM-01	Incorrect Error Message	Logical Issue	● Minor	✓ Resolved
HTM-02	Redundant Code	Logical Issue	● Informational	✓ Resolved
HTM-03	Contract Gains Non-Withdrawable HT Via The `swapAndLiquify` Function	Logical Issue	● Medium	ⓘ Acknowledged
HTM-04	Centralized Risk In `addLiquidity`	**Centralization / Privilege**	● **Major**	✓ **Resolved**
HTM-05	Variable Could Be Declared As `constant`	Gas Optimization	● Informational	✓ Resolved
HTM-06	Return Value Not Handled	Volatile Code	● Informational	ⓘ Acknowledged
HTM-07	3rd Party Dependencies	Control Flow	● Minor	ⓘ Acknowledged
HTM-08	Missing Event Emitting	Coding Style	● Informational	ⓘ Acknowledged
HTM-09	Function And Variable Naming Doesn't Match The Operating Environment	Coding Style	● Informational	✓ Resolved
HTM-10	Privileged Ownership	**Centralization / Privilege**	● **Minor**	ⓘ **Acknowledged**
HTM-11	Typos In The Contract	Coding Style	● Informational	✓ Resolved
HTM-12	The Purpose Of Function `deliver`	Control Flow	● Informational	ⓘ Acknowledged

ID	Title	Category	Severity	Status
HTM-13	Possible To Gain Ownership After Renouncing The Contract Ownership	**Logical Issue, Centralization / Privilege**	● **Minor**	ⓘ **Acknowledged**

HTM-01 | Incorrect Error Message

Category	Severity	Location	Status
Logical Issue	● Minor	HTMoon.sol: 872	✓ Resolved

Description

The error message in `require(_isExcluded[account], "Account is already excluded")` does not describe the error correctly.

Recommendation

The message "Account is already excluded" can be changed to "Account is not excluded" .

Alleviation

The client heeded the advice and resolved in commit `f4a442f83641fdaa1f65bec0512c522b90590db5`.

HTM-02 | Redundant Code

Category	Severity	Location	Status
Logical Issue	● Informational	HTMoon.sol: 1126	✓ Resolved

Description

The condition `!_isExcluded[sender] && !_isExcluded[recipient]` can be included in `else` .

Recommendation

The following code can be removed:

```
... else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
    _transferStandard(sender, recipient, amount);
} ...
```

Alleviation

The client heeded the advice and resolved in commit `f4a442f83641fdaa1f65bec0512c522b90590db5`.

HTM-03 | Contract Gains Non-Withdrawable HT Via The `swapAndLiquify` Function

Category	Severity	Location	Status
Logical Issue	● Medium	HTMoon.sol: 1061	ⓘ Acknowledged

Description

The `swapAndLiquify` function converts half of the `contractTokenBalance` HTMoon tokens to HT. The other half of HTMoon tokens and part of the converted HT are deposited into the HTMoon-HT pool on Uniswap as liquidity. For every `swapAndLiquify` function call, a small amount of HT leftover in the contract. This is because the price of HTMoon drops after swapping the first half of HTMoon tokens into HTs, and the other half of HTMoon tokens require less than the converted HT to be paired with it when adding liquidity. The contract doesn't appear to provide a way to withdraw those HT, and they will be locked in the contract forever.

Recommendation

It's not ideal that more and more HT are locked into the contract over time. The simplest solution is to add a `withdraw` function in the contract to withdraw HT. Other approaches that benefit the HTMoon token holders can be:

- Distribute HT to HTMoon token holders proportional to the amount of token they hold.
- Use leftover HT to buy back HTMoon tokens from the market to increase the price of HTMoon.

Alleviation

No Alleviation.

HTM-04 | Centralized Risk In `addLiquidity`

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	HTMoon.sol: 1108	✓ **Resolved**

Description

```
// add the liquidity
uniswapV2Router.addLiquidityETH{value: ethAmount}(
    address(this),
    tokenAmount,
    0, // slippage is unavoidable
    0, // slippage is unavoidable
    owner(),
    block.timestamp
);
```

The `addLiquidity` function calls the `uniswapV2Router.addLiquidityETH` function with the `to` address specified as `owner()` for acquiring the generated LP tokens from the `HTMoon-HT` pool. As a result, over time the `_owner` address will accumulate a significant portion of LP tokens. If the `_owner` is an EOA (Externally Owned Account), mishandling of its private key can have devastating consequences to the project as a whole.

Recommendation

We advise the `to` address of the `uniswapV2Router.addLiquidityETH` function call to be replaced by the contract itself, i.e. `address(this)`, and to restrict the management of the LP tokens within the scope of the contract's business logic. This will also protect the LP tokens from being stolen if the `_owner` account is compromised. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract based accounts with enhanced security practices, f.e. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent single point of failure due to the private key;
- Introduction of a DAO / governance / voting module to increase transparency and user involvement.

Alleviation

The client heeded the advice and resolved in commit `f4a442f83641fdaa1f65bec0512c522b90590db5`.

HTM-05 | Variable Could Be Declared As `constant`

Category	Severity	Location	Status
Gas Optimization	● Informational	HTMoon.sol	✓ Resolved

Description

Variables `_tTotal`, `numTokensSellToAddToLiquidity`, `_name`, `_symbol` and `_decimals` could be declared as `constant` since these state variables are never to be changed.

Recommendation

We recommend declaring those variables as `constant`.

Alleviation

The client heeded the advice and resolved in commit `f4a442f83641fdaa1f65bec0512c522b90590db5`.

HTM-06 | Return Value Not Handled

Category	Severity	Location	Status
Volatile Code	● Informational	HTMoon.sol: 1107~1114	ⓘ Acknowledged

Description

The return values of function `addLiquidityETH` are not properly handled.

```
        uniswapV2Router.addLiquidityETH{value: ethAmount}(
            address(this),
            tokenAmount,
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            owner(),
            block.timestamp
        );
```

Recommendation

We recommend using variables to receive the return value of the functions mentioned above and handle both success and failure cases if needed by the business logic.

Alleviation

No Alleviation.

HTM-07 | 3rd Party Dependencies

Category	Severity	Location	Status
Control Flow	● Minor	HTMoon.sol	ⓘ Acknowledged

Description

The contract is serving as the underlying entity to interact with third party Uniswap protocols. The scope of the audit would treat those 3rd party entities as black boxes and assume its functional correctness. However in the real world, 3rd parties may be compromised that led to assets lost or stolen.

Recommendation

We understand that the business logic of the HTMoon protocol requires the interaction Uniswap protocol for adding liquidity to HTMoon-HT pool and swap tokens. We encourage the team to constantly monitor the statuses of those 3rd parties to mitigate the side effects when unexpected activities are observed.

Alleviation

No Alleviation.

HTM-08 | Missing Event Emitting

Category	Severity	Location	Status
Coding Style	● Informational	HTMoon.sol	ⓘ Acknowledged

Description

In contract `HTMoon`, there are a bunch of functions can change state variables. However, these function do not emit event to pass the changes out of chain.

Recommendation

Recommend emitting events, for all the essential state variables that are possible to be changed during runtime.

Alleviation

No Alleviation.

HTM-09 | Function And Variable Naming Doesn't Match The Operating Environment

Category	Severity	Location	Status
Coding Style	● Informational	HTMoon.sol	✓ Resolved

Description

Function swapTokensForEth(uint256 tokenAmount) swaps HTMoon token for HT instead of ETH.

Recommendation

Change "ETH" to "HT" in the contract respectively to match the operating environment and avoid confusion.

Alleviation

The client heeded the advice and resolved in commit `f4a442f83641fdaa1f65bec0512c522b90590db5`.

HTM-10 | Privileged Ownership

Category	Severity	Location	Status
Centralization / Privilege	● Minor	HTMoon.sol	ⓘ Acknowledged

Description

The owner of contract `HTMoon` has the permission to:

1. change the address that can receive LP tokens,
2. lock the contract,
3. exclude/include addresses from rewards/fees,
4. set `taxFee`, `liquidityFee` and `_maxTxAmount`,
5. enable `swapAndLiquifyEnabled`

without obtaining the consensus of the community.

Recommendation

Renounce ownership when it is the right timing, or gradually migrate to a timelock plus multisig governing procedure and let the community monitor in respect of transparency considerations.

Alleviation

No Alleviation.

HTM-11 | Typos In The Contract

Category	Severity	Location	Status
Coding Style	● Informational	HTMoon.sol: 750, 922	✓ Resolved

Description

There are several typos in the code and comments.

1. In the following code snippet, `tokensIntoLiqudity` should be `tokensIntoLiquidity`.

```
event SwapAndLiquify(
        uint256 tokensSwapped,
        uint256 ethReceived,
        uint256 tokensIntoLiqudity
    );
```

2. `recieve` should be `receive` and `swaping` should be `swapping` in the line of comment `//to recieve ETH from uniswapV2Router when swaping`.

Recommendation

We recommend correcting all typos in the contract.

Alleviation

The client heeded the advice and resolved in commit `f4a442f83641fdaa1f65bec0512c522b90590db5`.

HTM-12 | The Purpose Of Function `deliver`

Category	Severity	Location	Status
Control Flow	● Informational	HTMoon.sol	ⓘ Acknowledged

Description

The function `deliver` can be called by anyone. It accepts an uint256 number parameter `tAmount`. The function reduces the HTMoon token balance of the caller by `rAmount`, which is `tAmount` reduces the transaction fee. Then, the function adds `tAmount` to variable `_tFeeTotal`, which represents the contract's total transaction fee. We wish the team could explain more on the purpose of having such functionality.

Alleviation

No Alleviation.

HTM-13 | Possible To Gain Ownership After Renouncing The Contract Ownership

Category	Severity	Location	Status
Logical Issue, Centralization / Privilege	● **Minor**	HTMoon.sol	ⓘ **Acknowledged**

Description

An owner is possible to gain ownership of the contract even if he calls function `renounceOwnership` to renounce the ownership. This can be achieved by performing the following operations:

1. Call `lock` to lock the contract. The variable `_previousOwner` is set to the current owner.
2. Call `unlock` to unlock the contract.
3. Call `renounceOwnership` to leave the contract without an owner.
4. Call `unlock` to regain ownership.

Recommendation

We advise updating/removing `lock` and `unlock` functions in the contract; or removing the `renounceOwnership` if such a privilege retains at the protocol level. If timelock functionality could be introduced, we recommend using the implementation of Compound finance as reference. Reference: https://github.com/compound-finance/compound-protocol/blob/master/contracts/Timelock.sol

Alleviation

No Alleviation.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we’re able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.