CERTIK

Security Assessment

Hoge Finance

Apr 25th, 2021

Summary

This report has been prepared for Hoge Finance 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 Dynamic Analysis, 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	Hoge Finance
Description	HOGE is a deflationary token. Each transaction takes place with HOGE, 2% of that transaction is distributed and burned from the total supply
Platform	Ethereum
Language	Solidity
Codebase	https://etherscan.io/address/0xfad45e47083e4607302aa43c65fb3106f1cd7607#code
Commits	hoge-finance

Audit Summary

Delivery Date	Apr 25, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Total Issues	5
Critical	0
Major	0
Minor	2
Informational	3
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
TCK	hogeToken.sol	4f419fa3918e30f310df2591fa3d06653eb7e53c91bd7d6c25d6d0951c4fc878

Centralization

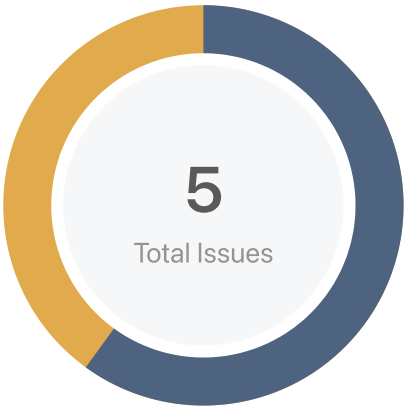
This is a deflationary token smart contract. The `onlyOwner` address had authority to include/ exclude address by functions:

- `includeAccount`
- `excludeAccount`

The advantage of the above functions in the codebase is that the client reserves the ability to adjust the project according to the runtime require to best serve the community. It is also worthy of note the potential drawbacks of these functions, which should be clearly stated through client's action/plan on how to prevent abuse of the these functionalities

To improve the trustworthiness of the project, any dynamic runtime updates in the project should be notified to the community. Any plan to implement aforementioned functions must be also considered to adopt Timelock with reasonable delay to allow the user to withdraw their funds, Multisig with community-selected 3-party independent co-signers, and/or DAO with transparent governance with the project's community in the project to manage sensitive role accesses.

Findings



Critical	0 (0.00%)
Major	0 (0.00%)
Minor	2 (40.00%)
Informational	3 (60.00%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
TCK-01	Redundant Code	Logical Issue	Informational	Acknowledged
TCK-02	Incorrect Error Message	Logical Issue	Minor	Acknowledged
TCK-03	Dynamic Rate Between `rSupply` and `tSupply`	Logical Issue	Informational	Acknowledged
TCK-04	Proper Usage of `public` and `external` type	Gas Optimization	Informational	Acknowledged
TCK-05	Centralized Risk	**Centralization / Privilege**	**Minor**	**Acknowledged**

TCK-01 | Redundant Code

Category	Severity	Location	Status
Logical Issue	Informational	hogeToken.sol: 580~582	Acknowledged

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

[Hoge]: The team acknowledged the issue, but can not make any changes in the current version.

TCK-02 | Incorrect Error Message

Category	Severity	Location	Status
Logical Issue	Minor	hogeToken.sol: 552	Acknowledged

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

[Hoge]: The team acknowledged the issue, but can not make any changes in the current version.

TCK-03 | Dynamic Rate Between `rSupply` and `tSupply`

Category	Severity	Location	Status
Logical Issue	Informational	hogeToken.sol: 552	Acknowledged

Description

Suppose the initial total supplies `_tTotal` = T_0^t and `_rTotal` T_0^r, then the initial exchange rate between `rSupply` and `tSupply` $r_0 = T_0^r / T_0^t$. After we make the first transfer of amount `x` from the initial owner to account `A`, the `r` balance of `A` `_rOwned[A]` $= O_A^r = 0.99x$. And `_rTotal` becomes $T_0^r - 0.01x$ because of the transfer fees. Then we exclude account `A` such that the `t` balance `_tOwned[A]` $= O_A^r / r_0 = 0.99x T_0^t / T_0^r$. Now the rate

$$r_1 = \frac{T_0^r - 0.01x - 0.99x}{T_0^t - 0.99x T_0^t / T_0^r} = \frac{T_0^r - x}{T_0^r - 0.99x} \cdot \frac{T_0^r}{T_0^t} < r_0$$

Similarly we can find the exchange rate will decrease as more accounts are excluded. However, as long as the majority of the supply is not excluded, the decrease will be small.

Alleviation

[Hoge]: The team acknowledged the issue, but can not make any changes in the current version.

TCK-04 | Proper Usage of `public` and `external` type

Category	Severity	Location	Status
Gas Optimization	Informational	hogeToken.sol: 457, 461, 465, 469, 473, 478, 483, 487, 492, 498, 503, 508, 512, 516, 525	Acknowledged

Description

Public functions that are never called by the contract could be declared external. When the inputs are arrays external functions are more efficient than `public` functions. Public functions that are never called by the contract could be declared external. When the inputs are arrays external functions are more efficient than `public` functions.

Example functions :

- name()
- symbol()
- totalSupply()
- balanceOf(address)
- transfer(address,uint256)
- allowance(address,address)
- approve(address,uint256)
- transferFrom(address,address,uint256)
- increaseAllowance(address,uint256)
- decreaseAllowance(address,uint256)
- isExcluded(address)
- totalFees()
- reflect(uint256)
- reflectionFromToken(uint256,bool)

Recommendation

Consider using the external attribute for functions never called from the contract.

Alleviation

[Hoge]: The team acknowledged the issue, but can not make any changes in the current version.

TCK-05 | Centralized Risk

Category	Severity	Location	Status
Centralization / Privilege	Minor	hogeToken.sol: 542, 551	Acknowledged

Description

`onlyOwner` address had authority to following functions:

- `excludeAccount()`
- `includeAccount()`

Recommendation

We advise the client to carefully manage the project's private key and avoid any potential risks of being hacked. We also advise the client to adopt Timelock with reason delay to allow the user to withdraw their funds, Multisig with community-selected 3-party independent co-signers, and/or DAO with transparent governance with the project's community in the project to manage sensitive role accesses.

Alleviation

[Hoge Finance]: This function exists to allow the exclusion of centralized exchanges from receiving redistribution as several exchanges are unable to integrate our tokenomics. Written approval of permission to exclude is received from the exchanges and shared with the community publicly. The development team voted on 3 trusted developers to control the smart contract via a multi-sig wallet. Note: the original developers who launched HOGE do not have access to the multi-sig wallet.

While we agree that this is a risk, we do not believe it is a major one given the steps we have taken and the nature of the internal workings of centralized exchanges.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that 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 exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that 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 and 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.

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.