HashEx

Barbershop Finance

smart contracts audit report

Prepared for:

barbershop.finance

Authors: HashEx audit team

August 2021

Contents

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report.

The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

Introduction

HashEx was commissioned by the Barbershop Finance team to perform an audit of their smart contracts. The audit was conducted between July 12 and July 20, 2021.

The code located in the github repository @barbershopfinance/barbershop-salon was audited after the commit 19892e7. The code was provided without any documentation.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts.
- Formally check the logic behind given smart contracts.

Information in this report should be used to understand the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts, by remediating the issues that were identified.

Update: Barbershop team has responded to this report. Individual responses were added after each item in the section. The updated code is located in the same repository after the 8a3cc16 commit. Updated contracts are deployed to the Polygon (MATIC):
`0x100A947f51fA3F1dcdF97f3aE507A72603cAE63C` HairToken,
`0xC6Ae34172bB4fC40c49C3f53badEbcE3Bb8E6430` Barber,
`0x76994C8364B5894b41048e1B2b2bC43847C19F3c` Timelock,
`0xd452C5A6Fc3d259D263C1480cfF6d43B7B1CD46c` HairVault.
At the time of writing the report, ownership of the HairToken was not transferred to the Timelock contract. Owners of the Barber and HairVault contract are externally owned accounts (EOA).

Contracts overview

HairToken.sol

A copy of SushiToken [1]. Implementation of ERC20 token standard with the minting open for the owner and governance from Yam Finance [2] (fork of Compound Governance Alpha, audit available [3]).

Barber.sol

The Staking contract is similar to MasterChef by SushiSwap [4] (the audit of which is available [5,6]) with minor modifications.

HairVault.sol

Farm contract to reinvest tokens into Staking contract, similar to CakeVault.

Timelock.sol

Timelock contract is similar to Timelock by Compound Finance [7] (audit available [8]) with minor modifications.

Found issues

ID	Title	Severity	Response
01	HairToken: delegates not transferred	High	Fixed
02	Barber: emission rate isn't limited	High	Fixed
03	Timelock: short minimum delay	Medium	Fixed
04	HairToken: mint() is open for owner	Medium	Responded
05	Barber: BONUS_MULTIPLIER not used	Low	Fixed
06	HairVault: excessive requirements in notContract()	Low	Fixed
07	HairVault: Pause and Unpause events are duplicated	Low	Fixed
08	HairVault: userInfo variables not used	Low	Responded
09	HairVault: isContract() function is duplicated	Low	Fixed
10	General recommendations	Low	Fixed

#01 HairToken: delegates not transferred — High

`_moveDelegates()` function in L207 of HairToken contract is designed to be used with every token transfer. However, in HairToken it's called with minting new tokens, but not with the usual transfers. The origin (SushiToken) has a warning (L8) about this issue. Issue allows to mint any number of delegation votes. A test for minting delegation votes can be seen in Appendix C.

Recommendation: stick with the original governance logic and move delegates with transfers.

Update: the issue was fixed.

#02 Barber: emission rate isn't limited — High

`updateEmissionRate()` function is used for updating the `hairPerBlock` parameter. Although it's the onlyOwner function, we recommend adding safety guards — capping the new value. If the owner's account gets compromised or the owner acts maliciously, the attacker can set an arbitrary big value for the `hairPerBlock` variable. In such a case the number of tokens till cap will be minted soon and the token price will drop. Moreover, it may become irreversible if an attacker adds new pools to the block gas limit of the `massUpdatePools()` function.

Recommendation: limit the `hairPerBlock` parameter in `updateEmissionRate()` function.

Update: the issue was fixed.

#03 Timelock: short minimum delay — Medium

`MINIMUM_DELAY` constant is set to only 6 hours which in our opinion is too small. Moreover, the first admin change doesn't need any delay, see L82.

Recommendation: we recommend changing the minimum delay of the contract to at least 12 hours or even better to transfer ownership to a new timelock with hardcoded `MINIMUM_DELAY` to 12 hours.

Update: the issue was fixed.

#04 HairToken: mint() is open for owner — Low

`mint()` function at L23 is open for the owner. In case of token redeployment, ownership should be transferred to the Barber contract as soon as possible. Users should check token ownership before using the token. It must be noted that once the ownership is transferred to the Barber contract, it becomes the only minter and can't transfer ownership further.

Barbershop team response: we are transferring ownership of the HairToken to the Barber immediately after deployment.

#05 Barber: BONUS_MULTIPLIER isn't used Low

`BONUS_MULTIPLIER` constant is set to 1 and therefore is useless.

Recommendation: we recommend removing an unused variable from the contract.

Update: the issue was fixed.

#06 HairVault: excessive requirements in notContract() Low

`notContract()` modifier at L99 contains two different checks of caller address. The second one is stricter and overlaps the terms of the extcodesize checking.

Update: the issue was fixed.

#07 HairVault: Pause and Unpause events are duplicated Low

HairVault contract contains `Pause()` and `Unpause()` events L63 which are emitted at the same time as inherited from Pausable contract `Paused()` and `Unpaused()` events [9].

Update: the issue was fixed.

#08 HairVault: userInfo variables not used Low

`hairAtLastUserAction` and `lastUserActionTime` parameters of userInfo structs aren't used in any of the audited contracts. The approximate user balance could be calculated with existing view functions and `user.shares`.

Barbershop team response: we do read from this contract on the frontend and lookup the timestamp to determine when the withdrawal fee period has passed.

#09 HairVault: isContract() function is duplicated Low

`_isContract()` function at L362 duplicates the `isContract()` function of Address library imported with the SafeERC20.

Update: the issue was fixed.

#10 General recommendations Low

Function `nonDuplicated()` in the Barber contract compares the boolean variable with 'false'.

Functions `setFeeAddress(), setDevAddress()` in the Barber contract lack checks for zero input address.

Typo in the comment in Barber (L64).

Functions `add(), set(), deposit(), withdraw(), emergencyWithdraw(), setDevAddress(), setFeeAddress(), updateEmissionRate()` in the Barber contract and `mint()` in the HairToken contract should be declared external.

Update: issues were fixed.

Conclusion

2 high severity issues were found. The contracts are highly dependent on the owner's account. Users using the project have to trust the owner and that the owner's account is properly secured.

Audit includes recommendations on the code improving and preventing potential attacks.

Update: Barbershop team has responded to this report. Most of the issues were fixed including all the high ones. Updated contracts are located in the same repository after the 8a3cc16 commit. Updated contracts are deployed to the Polygon (MATIC):
`0x100A947f51fA3F1dcdF97f3aE507A72603cAE63C` HairToken,
`0xC6Ae34172bB4fC40c49C3f53badEbcE3Bb8E6430` Barber,
`0x76994C8364B5894b41048e1B2b2bC43847C19F3c` Timelock,
`0xd452C5A6Fc3d259D263C1480cfF6d43B7B1CD46c` HairVault.
At the time of writing the report, ownership of the HairToken was not transferred to the Timelock contract. Owners of the Barber and HairVault contract are externally owned accounts (EOA).

References

1. SushiToken on GitHub
2. YAMGovernance on github
3. Compound Alpha audit by OpenZeppelin
4. SushiSwap's MasterChef contract
5. SushiSwap audit by PeckShield
6. SushiSwap audit by Quantstamp
7. CompoundFinance's Timelock contract
8. Timelock audit by OpenZeppelin
9. Pausable contract by OpenZeppelin

Appendix A. Issues' severity classification

We consider an issue to be critical, if it may cause unlimited losses, or breaks the workflow of the contract, and could be easily triggered.

High severity issues may lead to limited losses or break interaction with users or other contracts under very specific conditions.

Medium severity issues do not cause the full loss of functionality but break the contract logic.

Low severity issues are typically nonoptimal code, unused variables, errors in messages. Usually, these issues do not need immediate reactions.

Appendix B. List of examined issue types

Business logic overview

Functionality checks

Following best practices

Access control and authorization

Reentrancy attacks

Front-run attacks

DoS with (unexpected) revert

DoS with block gas limit

Transaction-ordering dependence

ERC/BEP and other standards violation

Unchecked math

Implicit visibility levels

Excessive gas usage

Timestamp dependence

Forcibly sending ether to a contract

Weak sources of randomness

Shadowing state variables

Usage of deprecated code

Appendix C. Delegation votes minting test

The code of the test in hardhat framework:

Alice gets 100 tokens and generates voting power for Carol of 300 tokens.

```
describe('Delegation power attack', async function () {
  it('should multiply delegation power on transfers', async function()  {
    await hairToken.mint(deployer.address, 100);
    await hairToken.delegate(carol.address);
    await hairToken.transfer(alice.address, 100);

    await hairToken.connect(alice).delegate(carol.address);
    await hairToken.connect(alice).transfer(bob.address, 100);
    await hairToken.connect(bob).delegate(carol.address);
    const votes = await hairToken.getCurrentVotes(carol.address);
    console.log('votes', votes.toString())
  })
});
```

The test output (Carol gets voting power of 300):

```
  HairToken
    Delegation power attack
votes  300
      ✓ should multiply delegation power on transfers
```