SOLIDIFIED

Audit Report for Hodl on June 30th 2020.

Summary

Audit Report prepared by Solidified for Hodl covering the Hodl Commodity smart contracts (and their associated components).

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The debrief took place on June 30th, 2020, and the final results are presented here.

Audited Files

The following contracts were covered during the audit:

https://github.com/jpbeaudet/HODLCommodity

Notes

The audit was based on commit `510194c3a7d074277bc0406f3e6f35fbab540b02`, Solidity compiler version `0.6.6`. Follow up was based on commit `10408634d869c98a03d638a6ec7ec3867a1b546d`.

Intended Behavior

HodlDex is a decentralized exchange where users could buy and sell the HodlT token for ETH. ETH funds collected users buying HodlT from the contract reserves are to be distributed almost immediately (at least next block) to a pseudo-random HodlT token holder, via a lottery system.

SOLIDIFIED

Audit Report for Hodl on June 30th 2020.

Executive Summary

Solidified found that the Hodl's contracts contain seven issues, with one of them being critical and two of major severity.

We recommend all issues are amended, while the notes are up to Hodl's discretion, as they mainly refer to improving the operation of the smart contracts and best practices.

Update [07.07.2020]
All issues were fixed and are no longer present in commit `10408634d869c98a03d638a6ec7ec3867a1b546d`.

Summary of Issues found

Critical	Major	Minor	Notes
1	3	4	6

SOLIDIFIED

Audit Report for Hodl on June 30th 2020.

Issues Found

Critical Issues

1. HodlDex.sol: The distribute modifier is vulnerable to Sybil attacks

---

Since the distribute modifier gives all hodlerAddrSet accounts an equal chance of winning the distribution lottery, an attacker can create a substantially large number of accounts holding minimum amounts of HodlT tokens in order to guarantee a very high chance of winning the lottery.

Recommendation
Consider making the chance of winning the distribution lottery directly proportional to the amount of HodlT tokens each account owns.

Another possible way of mitigating this is requiring a minimum value held by the user (as it makes the attack more expensive).

Amended [07.07.2020]
The issue was fixed and is no longer present in commit `10408634d869c98a03d638a6ec7ec3867a1b546d`.

Major Issues

2. HodlDex.sol: Pseudo randomness for selecting winner of distribution bonus can be influenced

---

In the modifier distribute, the keccak hash of the balance of the contract is used as the source of entropy. The contract will then perform a modulo operation to select the winner.

The calculation is easily replicable as all parameters (the contract's hodl balance and the number of users) are always known.

A dishonest player can "mine" a perfect amount that will make him the winner of the next distribution and submit a transaction with high fees to ensure his transaction is mined just before the end of the distribution period.

Recommendation
The documentation mentions the randomness source is pseudo-random, meaning the intent was not to create a flawless random number generation. The current algo can, however, be improved by including other sources of entropy. Including the timestamp (miner controlled), msg.sender balance (caller controlled), read() from maker oracle (dependent on USD ETH price at the time and a number of actors), and the current parameters (internal from the contract, dependent on the time the distribution happens) will increase the time and effort to guess the draw, and reduce certainty that an attack will succeed.

Amended [07.07.2020]
The issue was fixed and is no longer present in commit `10408634d869c98a03d638a6ec7ec3867a1b546d`.

3. HodlDex.sol: Incorrect accrual

The bug found creates a discrepancy in the accounting for the 'accrue by transaction' operation. Each action that buys tokens from a sell order should create an increment of $0.0001 per transaction. The expected results should be that if a buy order consumes 1 sell order completely and 1 sell order partially we would expect and increment of $0.0001 per sell order that the buy order interacts with. In the above example the total increase should be $0.0002. The bug creates, for the above example, a total increase of $0.0003. The reason this happens is because memory variable `orderHodl` is used for a logic check and is not updated when the storage variable `o.volumeHodl` is modified and the loop goes around one more time calling `_accrueByTransaction()` an extra time, generating the extra increase of $0.0001.

* This bug was found and reported to us by the Hodl team.

Recommendation

Change line 313:
if(orderHodl == 0) {

To:
if(o.volumeHodl == 0) {

Amended [07.07.2020]
The issue was fixed and is no longer present in commit `10408634d869c98a03d638a6ec7ec3867a1b546d`.

4. HodlDex.sol: Function orderLimit() returns the opposite of intended behavior

---

According to documentation found in comments, the contract should remove order limits once HODL_USD is greater than maxThreshold. However, the function orderLimit() does the opposite, removing limits when HODL_USD is less than or equal to maxThreshold, and otherwise returning maxOrderUsd (thus limiting order sizes when HODL_USD is greater than maxThreshold).

Recommendation
Change line #518:
`return (askUsd > maxThreshold) ? maxOrderUsd : 0;`

To:
`return (askUsd > maxThreshold) ? 0 : maxOrderUsd;`

Amended [07.07.2020]
The issue was fixed and is no longer present in commit `10408634d869c98a03d638a6ec7ec3867a1b546d`.

Minor Issues

5. HodlDex.sol: There is no way to externally call distribute and accrueByTime in case no users interact with HodlDex

---

Currently, the only way to externally call distribute or accrueByTime is for HodlDex to remain active with user transactions. If no users interact with any of HodlDex state modifying functions (for several days, say), these essential contract plumbing calls will not take place.

Recommendation
Create a dedicated function that allows anyone to externally call distribute and accrueByTime. A Better solution is to adjust the `_accrueByTime` function to take care of all unprocessed days at once.

Sample code:
```
  function poke() external distribute ifRunning {
      _accrueByTime();
  }
```

Amended [07.07.2020]
The issue was fixed and is no longer present in commit `10408634d869c98a03d638a6ec7ec3867a1b546d`.

6. HodlDex.sol: The contract's HodlT balance is not being updated in initResetUser()

In initResetUser, the contract's balance is not being updated. Furthermore, there is a typo where r.balanceHodl is being added instead of u.balanceHodl.

```
r.balanceHodl.add(r.balanceHodl);
```

The line above should read like the following

```
r.balanceHodl = r.balanceHodl.add(u.balanceHodl)
```

Recommendation
Correct the function as described in the issue description. Also consider a batch input for users (initUser) as it will save a considerable amount of gas from sending one transaction per user.

Amended [07.07.2020]
The issue was fixed and is no longer present in commit `10408634d869c98a03d638a6ec7ec3867a1b546d`.

7. HodlDex.sol: Some holders might stay out of distribution or non-holders might be included.

It's unclear whether this is the intended behavior, but users who issue or redeem through `hodlTIssue` and `hodlTRedeem` aren't properly included or pruned from the address Set.

Recommendation
Amend the aforementioned functions to call `makeHodler` and `pruneHodler`, if necessary.

Amended [07.07.2020]
The issue was fixed and is no longer present in commit `10408634d869c98a03d638a6ec7ec3867a1b546d`.

8. HodlDex might become illiquid

There's an implicit invariant that the DEX contract always owns enough tokens to cover issuance, but that's not necessarily true, because this action has to be done separately, as the token's initial supply is minted to the transaction msg.sender.

Recommendation
Consider updating the HodlDEX balance with an external call to the token, before trading starts, or at least checking if the balance corresponds to the expected value (the totalSupply).

Amended [07.07.2020]
The issue was fixed and is no longer present in commit `10408634d869c98a03d638a6ec7ec3867a1b546d`.

Notes

9. HodlDex.sol: Function rates() includes redundant calls to both isAccruing() and now > BIRTHDAY.add(SLEEP_TIME)

Since the isAccruing() function already evaluates to now > BIRTHDAY.add(SLEEP_TIME), there is no need to do this check a second time on line #426.

Amended [07.07.2020]
The issue was fixed and is no longer present in commit `10408634d869c98a03d638a6ec7ec3867a1b546d`.

10. HodlDex.sol: No need for using the SafeERC20 library

The safeTransfer() / safeTransferFrom() functions are typically only needed when interacting with tokens that might not return a value for transfer() / transferFrom(). Since the HodlT token is based on OpenZeppelin's ERC20 contract, it properly returns values for transfer() & transferFrom(), and hence does not require the use of SafeERC20.

Recommendation
Remove the unneeded SafeERC20 library to reduce code complexity.

Amended [07.07.2020]
The issue was fixed and is no longer present in commit `10408634d869c98a03d638a6ec7ec3867a1b546d`.

11. HodlDex.sol: Consider commenting out unused variable names to suppress compiler warnings

For instance, replace the following:
(uint askUsd, uint accrualRate) = rates();
accrualRate;

With:
`(uint askUsd, /*uint accrualRate*/) = rates();`

Amended [07.07.2020]
The issue was fixed and is no longer present in commit `10408634d869c98a03d638a6ec7ec3867a1b546d`.

12. FIFOSet.sol: When appending the initial key, function append() redundantly sets self.keyStructs[NULL].nextKey to key

When the very first key is appended, `lastKey` is equal to `NULL`, yet `self.keyStructs[lastKey].nextKey` is still being set. Since `self.keyStructs[NULL].nextKey` can never be accessed by the user in any of the library's functions, this assignment becomes an unnecessary allocation of storage.

Recommendation
Only set `self.keyStructs[lastKey].nextKey` if `lastKey` does not equal `NULL`.

Amended [07.07.2020]
The issue was fixed and is no longer present in commit `10408634d869c98a03d638a6ec7ec3867a1b546d`.

13. FIFOSet.sol: Consider prepending error messages with library name

In order to stay consistent with error messages in `AddressSet` and `Bytes32Set`, consider perpending error message strings in the `FIFOSet` library with "`FIFOSet: `".

Amended [07.07.2020]
The issue was fixed and is no longer present in commit `10408634d869c98a03d638a6ec7ec3867a1b546d`.

14. FIFOSet.sol: Consider refactoring test contract FIFO into its own file

Consider moving the `FIFO` test contract into its own file for better code readability.

Amended [07.07.2020]
The issue was fixed and is no longer present in commit `10408634d869c98a03d638a6ec7ec3867a1b546d`.

Audit Report for Hodl on June 30th 2020.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of the Hodl platform or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.