SOLIDIFIED

Audit Report for Hodl on November 05, 2020

Summary

Audit Report prepared by Solidified for Hodl covering version 2 of the Hodl Commodity smart contracts (and their associated components).

Process and Delivery

Four (4) independent Solidified experts performed an unbiased and isolated audit of the code below. The debrief took place on October 15, 2020.

Additional interaction with the team, further reviews and fixes took place through November 5, 2020.

Audited Files

The following contracts were covered during the audit:

```
contracts
├── HOracle.sol
├── HProxy.sol
├── HTokenReserveProxy.sol
├── HTokenReserveProxyAdmin.sol
├── HodlDex.sol
├── HodlDexProxy.sol
├── HodlDexProxyAdmin.sol
├── Migrations.sol
├── ProportionalTest.sol
├── libraries
│   ├── AddressSet.sol
│   ├── Bytes32Set.sol
│   ├── FIFOSet.sol
│   └── Proportional.sol
└── token
    ├── HTEthUsd.sol
    └── HTokenReserve.sol
```

The contracts were supplied in the repositories:

https://github.com/HODLCommodity/HODLDex2

SOLIDIFIED

Audit Report for Hodl on November 05, 2020

Notes

The audit was based on commit `eeff2fe3ac9ab95fda9cb68fc3bce1c8a4fba4da`.

Updates were provided in several commits culminating in commit `941ab6fb40fc8c4b4ce95f5866618f6367812249`

Intended Behavior

The smart contracts implement an algorithmically priced token that can be sold and bought with ETH using a contract implementing a unique order book.
The price is increased on a timebase algorithm and on each buy order (HC). A separate commodity (HT) is pegged to the USD. The DEX facilitates conversion between the two tokens.

SOLIDIFIED

Audit Report for Hodl on November 05, 2020

Executive Summary

Solidified found that the Hodl contracts contain 12 issues and 5 informational notes.

We recommend all issues are amended, while the notes are up to the team's discretion, as they refer to best practices or optimizations.

Given the complexity of the codebase, we encourage a pre-launch bug bounty to be held.

Issues found:

Issue #	Description	Severity	Status
1	HOracle.sol: Bug in read() function returns the wrong ethUsd18 rate	Critical	Resolved
2	HodlDex.sol: HodlDex's admin can obtain HCETHUSD at close to zero cost	Major	Acknowledged
3	HTokenReserve.sol: HTokenReserve's admin can issue themselves an unlimited amount of HTETHUSD	Major	Acknowledged
4	HodlDex.sol: Redeeming HTETHUSD will always fail due to calling a non existent HTokenReserve function	Major	Resolved
5	HodlDex.sol: Function initResetUser() does not reset back the user's HCETHUSD balance	Major	Resolved
6	Admin has full control of storage	Major	Acknowledged
6a	SellOrder.askUsd is never used after setting it	Major	Resolved
7	Automated Surplus Allocation will only be called through modifier on block numbers that are exact multiples of 500	Minor	Resolved

8	Ether transfers depending on gas stipend may not work if the recipient is a smart contract	Minor	Resolved
9	HodlDex.sol: Incorrect initialization of accrualDaysProcessed	Minor	Resolved
10	HodlDex.sol: an account with the ORACLE_ROLE role can unnecessarily increment transaction count	Minor	Acknowledged
11	HodlDex.sol: Accrue by time function can have unprocessed days	Minor	Acknowledged
12	HodlDex.sol: cancelSell() and cancelBuy() functions do not delete cancelled orders from their respective mappings	Note	Resolved
13	Different version pragmas	Note	Resolved
14	Proportional.sol: possible overflow on add() function	Note	Resolved
15	Proportional.sol: possible overflow on add() function	Note	Resolved
16	HodlDex.sol: Duplicate call of balance.poke()	Note	Resolved

SOLIDIFIED

Audit Report for Hodl on November 05, 2020

Critical Issues

1. HOracle.sol: Bug in read() function returns the wrong ethUsd18 rate

---

Function `read()` always returns the wrong `ethUsd18` rate due to a conversion bug.

Recommendation
Change line #21 from:

```
ethUsd18 = (uint(uniswapReserve0) * PRECISION) / (uniswapReserve1 * UNISWAP_SHIFT);
```

To:

```
ethUsd18 = (uint(uniswapReserve0) * PRECISION * UNISWAP_SHIFT) / (uniswapReserve1);
```

Update - Fixed!

Major Issues

2. HodlDex.sol: HodlDex's admin can obtain HCETHUSD at close to zero cost

With its current implementation, `HodlDex`'s admin can temporarily provide a malicious Oracle via the `adminSetOracle()` function. This would potentially allow them to obtain `HCETHUSD` at close to zero cost by having the new oracle report substantially high ETH prices.

In addition, function `oracleSetEthUsd()` also allows for a very similar attack by any account with the `ORACLE_ROLE` role.

Recommendation
Provide a permanent oracle that can only be changed by upgrading the `HodlDex` contract, and remove the `ORACLE_ROLE` role entirely (`getEthToUsd()` is sufficient for querying the oracle). The assumption here is that the `HodlDexProxy` contract's admin will be a governance contract, where upgrades can only occur upon agreement by the majority of `HCETHUSD` holders.

Update - Team Response
"This capability mitigates a well-known risk when critical aspects of the system rely on external on-chain components.

In the case that our external dependency on Uniswap stops working as expected, we allow for the injection of a correct quote with 3 signers. This is intended as a bridge solution to support continued trading while a replacement HOracle contract is developed and deployed. "

3. HTokenReserve.sol: HTokenReserve's admin can issue themselves an unlimited amount of HTETHUSD

Since `HTokenReserve`'s admin is assigned the `DEFAULT_ADMIN_ROLE` role at initialization, they can potentially grant a new `DEX_ROLE` role to any account of their choice at any point in time. This would allow them to issue themselves an unlimited amount of `HTETHUSD` by calling the `issueHTEthUsd()` function with the newly assigned role.

Recommendation
Remove line #55:

```
_setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
```

There is no need for the `init()` caller to get the `DEFAULT_ADMIN_ROLE` role. In case roles need to be modified in the future, deploy a new contract upgrade with the approval of the majority of Hodl users (as previously discussed).

Update - Team Response
“We assign the DEFAULT_ROLE_ADMIN to the multi-signature contract to support migration away from the initial structure which is based on executive control, accountability and off-chain governance policy. For example, roles can be transferred to other contracts.

Additionally, an emergency stop is possible (with 3 signatures) by assigning an address to the MIGRATION_ROLE and this can be an address with an unknown signing key.

The signers will be legally restrained from colluding to defraud the community. “

4. HodlDex.sol: Redeeming HTETHUSD will always fail due to calling a non existent HTokenReserve function

---

The `hodlTRedeem()` function will always fail to redeem due to it calling a non existent function of `HTokenReserve`.

Recommendation
Change line #216 from:

```
uint amountHodl = tokenReserve.burnHTEthUsd(msg.sender, amountUsd);
```

To:

```
uint amountHodl = tokenReserve.redeemHTEthUsd(msg.sender, amountUsd);
```

Update - Fixed!

5. HodlDex.sol: Function initResetUser() does not reset back the user's HCETHUSD balance

The `initResetUser()` does not set the user's HCETHUSD balance back to zero, which results in returning the ETH funds without reclaiming the user's HCETHUSD tokens.

Recommendation
Reset the user's balance by adding the following statement:

```
balance.sub(HODL_ASSET, userAddr, balance.balanceOf(HODL_ASSET, userAddr),0);
```

Update - Fixed!

6. Admin has full control of storage

The Hodl admin can upgrade the contract at any time, which means they have access to all storage variables, including user balances. If this particular key gets compromised, the side effects could be catastrophic.

Recommendation
It's recommended to delegate this key to a multisig or even better, a proper governance mechanism to take care of protocol updates. Another important piece is to add a delay in upgrades, so users have time to react in case of an unwanted change.

Update - Team Response
"Agreed. The two ProxyAdmin contracts will be assigned to the multi-signature contract requiring 3 of 5 signatures. This will be verifiable post-deployment."

6a. SellOrder.askUsd is never used after setting it.

The field `askUsd` of struct `SellOrder` declared in `HodlDex.sol` is never used after setting it. This means that the management of sell orders does not match the specification. When

calculating the ETH price in _fillSellOrders(), _convertHodlToEth() is being called instead of using SellOrder.askUsd. This contradicts their specification that the sell order price does not increase with time.

Recommendation
Adapt behaviour to specification or revise specification.

Update - Fixed!

Minor Issues

7. Automated Surplus Allocation will only be called through modifier on block numbers that are exact multiples of 500

The `periodic` modifier in `HTokenReserve.sol` calls `allocateSurplus()` can when the current block number is a precise multiple of `FREQUENCY` (set to 500):

```
modifier periodic {
        if(block.number % FREQUENCY == 0) allocateSurplus();
        _;
}
```

This means that the surplus allocation may only be executed occasionally at infrequent intervals. Even though surplus allocation can be triggered manually through a public function, the modifier trigger will not work as it is likely to be intended.

Recommendation
This can be solved by keeping track of the last block number `allocateSurplus()` has been called and executing if 500 blocks or more have passed.

Update - Fixed!

8. Ether transfers depending on gas stipend may not work if the recipient is a smart contract

The `transfer()` function is used to transfer ETH in `HodlDex.sol` (line 753). With the Istanbul hard fork gas prices changed meaning the forwarded gas of 2300 may not be enough for smart contract recipients to perform their operations, leading to the call reverting.

Recommendation
It is now recommended to use `call.value()` for transferring ETH.

Update - Fixed!

9. HodlDex.sol: Incorrect initialization of accrualDaysProcessed

Storage variable `accrualDaysProcessed` is incorrectly initialized in function `init()`.

Recommendation
Change line #711 from:

```
accrualDaysProcessed = accrualDaysProcessed;
```

To:

```
accrualDaysProcessed = _accrualDaysProcessed;
```

Update - Fixed!

10. HodlDex.sol: an account with the ORACLE_ROLE role can unnecessarily increment transaction count

Function `increaseTransactionCount()` allows any account with the `ORACLE_ROLE` role to increment the transaction count. Since an oracle should in no way be creating any Hodl transactions, such a function is unnecessary and opens the door for transaction count manipulation by oracles.

Recommendation
Remove the `increaseTransactionCount()` function.

Update - Team Response
“The function exists to support communication from contracts on other chains which is part of the publicized project roadmap. The function is implemented to avoid a necessary upgrade for a scenario that is planned.

This function is not initially required and no account will have permission to use it until 3 signatures grant the role to an external entity. “

11. HodlDex.sol: Accrue by time function can have unprocessed days

If the contract is not called for more than a day, the next time it's called it won't consider the skipped days in the accrual calculation

Recommendation
Save the last time this function was executed so it always accrues correctly.

Update - Team Response
"This is by design. The contract presents correct rates via the view function, rates() and will continue to do so in the 24-48 hours period even though no gas is provided to update the internal state. Those rates are applied to the next transaction, correctly, and the next transaction advances the internal by 24 hours if needed.

We do not attempt to process multiple days simultaneously because:

There are seldom days without transaction activity
Iterating over days to process would be an unbounded loop
An exponent can solve the equation in one pass by raising the adjustment to the power of the days to process, but this can cause an overflow error when integers are multiplied.

Our solution provides eventual adjustment by processing up to one day per transaction, and always ensures that the advertised rate (view function) matches the rate that will be used for a transaction received now."

Notes

12. HodlDex.sol: cancelSell() and cancelBuy() functions do not delete cancelled orders from their respective mappings

Functions `cancelSell()` and `cancelBuy()` do not remove the cancelled orders from their `sellOrder` and `buyOrder` mappings, respectively. This results in unnecessary wasted storage space.

Recommendation
Remove orders from their respective mappings as soon as they're cancelled.

Update - Fixed!

13. Different version pragmas

The following contracts use an lower version pragma then the rest of the code:

`libraries/Bytes32Set.sol`
`libraries/AddressSet.sol`

Recommendation
It is recommended to enforce the same compiler version throughout the codebase.

Update - Fixed!

14. HodlDex.sol: modifier onlyReserve has the wrong require error message

The `onlyReserve` modifier's error message should read "HodlDex 403 reserve" instead of "HodlDex 403 migration".

Update - Fixed!

15. Proportional.sol: possible overflow on add() function

This function does not use SafeMath library and can potentially overflow, although that's unlikely. It may be an additional concern in case the oracle is compromised or faulty.

Recommendation
Use safeMath.

Update - Team Response
"We have optimized the contracts using SafeMath only where we identify a risk of overflow/underflow and we do not anticipate any scenario where overflow is possible, but we agree with the abundance of caution.

Since we are using SafeMath sparingly we encourage the auditors to highlight any other math operations where it is not impossible to overflow/underflow."

Update - Fixed!

16. HodlDex.sol: Duplicate call of balance.poke()

The public function `poke()` calls `_setEthToUsd()` and then immediately calls `balance.poke()`, which is also called at the end of `_setEthToUsd()`, resulting in two sequential identical calls.

Recommendation
Remove the last two lines of `poke()`.

Update - Fixed!

Audit Report for Hodl on November 05, 2020

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Hodl 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.