SOLIDIFIED

Audit Report for Loopring on May 21st, 2020.

Summary

Audit Report prepared by Solidified for Loopring covering the Hebao V1 Wallet 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 May 21st, 2020, and the final results are presented here.

Audited Files

Files contained in folder Hebao_V1, from the repository/commit listed in the next section. All smart contracts in the contracts folder are in scope with the exception of the third_pary folder.

Additionally, in the /lib directory, only the following files were in-scope: AddressSet.sol, EIP712.sol, Managable.sol, Mathint.sol, OwnerManagable.sol and SignatureUtil.sol, though other contracts imported by in-scope contracts were considered.

Notes

The audit was initially performed on commit `db7bf88f84a77270d7e6b67c3534d0d1eaedaacf` of repository https://github.com/Loopring/protocols, using Solidity compiler version `0.6.6`.

Intended Behavior

Hebao is a smart contract wallet that allows users to store their funds and interact with Loopring´s exchange and other DeFi Dapps through modules.

SOLIDIFIED

Audit Report for Loopring on May 21st, 2020.

Issues Found

Critical

1. BaseVault.execute(..) transactions can be replayed.

`BaseVault.execute(..)` function does not implement any signed transaction replay protection. For example, if the vault owners sign a value or token transfer transaction, the transaction could be repeated many times by anyone successfully as long as the vault has funds.

Recommendation
Implement a nonce or another form of replay protection, as implemented on the MetaTxModule.

2. Reentrancy vulnerability in MetaTxModule.sol, function executeTransactions()

All signing rules can be bypassed by a single guardian or owner.
To call executeTransactions on executeMetaTx, a single signature is required, the calls passed as a parameter could be used to reenter the executeTransactions function (passing through the onlyMetaTx modifier because the call is from the contract itself, and most importantly bypassing the signature checks on executeMetaTx) and send any transaction on behalf of the wallet, bypassing controls such as majority requirements, only owner adding modules or whitelist.

```
 function executeMetaTx(...)
       .....
        address[] memory signers = getSigners(wallet, data);
        require(areMetaTxSignersAuthorized(wallet, data, signers),
"METATX_UNAUTHORIZED");
function getSigners(..)
          ....
        bytes4 method = extractMethod(data);
        if (method == this.executeTransactions.selector) {
            return extractAddressesFromCallData(data, 1);
```

Though there is a check in the executeTransactions() that signers addresses match the required ones, it is possible to specify an arbitrary signers array parameter when doing a recursive call to executeTransactions().

Recommendation
Couple signature verification and transaction execution tightly, in order to prevent owners and controllers from bypassing wallet restrictions.

3. Mathint.sol does not prevent underflow/overflow.

The checks which are supposed to check overflow/underflow conditions do not work because they also overflow/underflow:

```
    int MAX_INT = 2^255 - 1
    int MIN_INT = - 2^255
    add(MAX_INT, 1) =
-57896044618658097711785492504343953926634992332820282019728792003956564819968
    add(MAX_INT-1, 2) =
-57896044618658097711785492504343953926634992332820282019728792003956564819968

    sub( MIN_INT, 1) =
57896044618658097711785492504343953926634992332820282019728792003956564819967
    sub( MIN_INT+1, 2) =
57896044618658097711785492504343953926634992332820282019728792003956564819967
```

Additionally, the `mul (int, int)` does not handle a single special case:

```
    mul(-1, MIN_INT) =
-57896044618658097711785492504343953926634992332820282019728792003956564819968
```

Recommendation

Use OpenZeppelin implementation: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/math/SignedSafeMath.sol.

Major

4. Deployer key holds a vast amount of power.

The deployer key has the ability to add managers to all contracts that inherit from `DataStore.sol`, which controls vital information from wallets, like locks, guardians, whitelist, etc. If this key gets compromised, an attacker might take ownership of all wallets, by adding himself as guardian, or inheritor, etc.

Having only one address in charge of ownership can have consequences to the process if the key is lost or worse, misappropriated.

Recommendation
Delegate the ownership to a multisig wallet controlled by several different Loopring stakeholders or to a governance structure, as this will prevent one key from being misused and enables recovery if any of the parties lose their keys.

Though not in-scope, during the audit we also noted the `ENSRegistry` is also controlled by a single address.

5. MetaTxModule.sol: Transactions without nonce are allowed

The contract implements a nonce for meta transactions, but it also allows transactions without one to be sent out.

The contract does prevent replays of transactions (by storing the tx hash and verifying it for new transactions), but the absence of a nonce will still enable a relayer to sort the user´s transactions deliberately, possibly allowing him to cause the transactions to fail by sending them on the wrong order, or censoring part of them.

Additionally, the absence of a nonce will prevent users from “burning” a lost transaction by sending another one with the same nonce, as those are valid indefinitely and can be relayed at any point in the future by the relayers.

Recommendation
Enforce the nonce for all externally signed transactions.

6. MathUint.decodeFloat() does not revert on overflow

If the exponent is higher than `0xFF`, the `10 **` exponent expression will return `0`.
If a supplied parameter is bigger than `0x7FFFFFFF` the function `decodeFloat()` will return `0`.

We classified this issue as major because it does not follow the convention of the `MathUint` library to revert on overflow and it is also not documented.

Recommendation
Ensure `decodeFloat` reverts on overflows.

7. Guardian and owner could be the same address

If a guardian either inherits a wallet or gets ownership by the recovery mechanism, the implicit assumption that the guardians are different then the owner will break. This might cause wrong permissions in calling functions, for example, lock wallet, as well cause miscalculations, or even prevent the wallet from certain actions in the `requireMajority` function.

Recommendation
Inheritor and Recovery modules should remove the new Owner from the Guardians list.

Minor

8. MetaTxModule.sol: Relayer can cause transactions to fail by sending just above the limit the user provided.

Due to the way the VM provides gas to external calls since EIP 150 ws implemented, if the external call is executed with less gas than the `gasSettings.limit`, only 63/64 of the available gas will be provided, allowing for the relayer to force transactions with a limit lower than the one set by the user.

A discussion around this issue is available at
https://github.com/gnosis/safe-contracts/issues/100

Recommendation
Require that the gas available after the call is greater than than 65/64 of the user provided gasSettings.limit.

For reference: Implementation of the fix by the Gnosis Safe team:
https://github.com/gnosis/safe-contracts/commit/62d4bd39925db65083b035115d6987772b2d2dca

9. CompoundModule.sol and Inheritance: Deposits and withdraws to Compound and change of Inheritance settings are allowed with a locked wallet.

Functions in the Compound and Inheritance modules are missing the modifier `onlyWalletUnlocked`, effectively allowing for Compound use from locked wallets.

Recommendation
Include the aforementioned modifier in the functions.

10. BaseSubAccount.sol: Function canDepositToken currently returning withdrawable amount.

Function `canDepositToken` is currently returning `tokenWithdrawable`, as is can `WithdrawToken`. Before any deposits are made, the function will always return 0.

Recommendation
Call `tokenDepositable` within the function instead.

11. BaseSubAccount.sol: Functions TokenReturnAmount and tokenWithdrawable do not implement functionality for ETH.

Functions `TokenReturnAmount` and `tokenWithdrawable` do not implement functionality for ETH. The implementation of `tokenDepositable` implies that ETH is accepted by passing Address(0x0) as the token address. Currently deposited Ether will not be withdrawable and will remain effectively frozen after deposit.

Recommendation
Ensure all deposit, withdraw and return functions implement ETH deposit/withdraw functionality.

12. ModuleRegistry.sol: Contract is owned by only one address.

ModuleRegistry, where new modules are registered to become available to users, Is controlled by only one owner. The owner account will be controlled by Loopring.

Having only one address in charge of ownership can have consequences to the process if the key is lost or worse, misappropriated.

Recommendation
Delegate the ownership to a multisig wallet controlled by several different Loopring stakeholders, as this will prevent one key from being misused and enables recovery if any of the parties lose their keys.

13. Consider using Uniswap V2 price oracle

It is possible to manipulate the token price on pooled-exchanges such as Uniswap and Kyber. Buying and selling tokens from/to such pools introduce a price slippage. It is dangerous to rely on a price from pulled exchanges especially when the token exchange pool is small - it takes smaller funds to manipulate the price. There have been such attacks carried out in the past - once the pay-off is larger than the funds needed to manipulate a pool.

The updateTokenPrice() function call could be front-run to manipulate the cached token price. The issue is classified as minor because currently only QuotaTransfers are using this functionality.

Recommendation
Use Uniswap V2, as well as increasing the number and nature of oracles.

14. Attacks by malicious guardians

If a wallet has only one guardian, he/she can take ownership of the wallet by itself.

A malicious guardian could also make the wallet hostage by constantly locking it. Similarly, a single guardian could also force the wallet to be unlocked while some kind of attack is performed.

Recommendation
There aren't many actions that can prevent this type of behaviour that also don't have side effects. Therefore, the best approach is to educate users of the pitfalls of guardians.

15. Dapp Modules can bypass quotas and whitelists

Most implemented Dapp modules have a deposit function, which doesn't take into account the defined spending quotas. The generic module could also be used to bypass token transfer quotas.

Recommendation
Dapps should also be subject to the limits imposed by transfer, whitelist and quota modules.

16. Vault can have more than MAX_OWNER owners

The vault only checks the size of the owners array in the constructor, but new owners could be added later, breaking the implied invariant of max owners.

Recommendation
The function `addOwner` should check for the `MAX_OWENRS` constant.

Notes

17. MetaTxModule.sol, ApprovedTransfers.sol: Return value of operations in collectTokens and reimburseGasFee is not being verified

In `collectTokens`, the return value of `sendEth` and of the `ERC20` transfers are not verified. The function will succeed if any of the transfer fails. This is also the case with `reimburseGasFee` (ERC20 tokens only).

On `TransferModule.sol`, The transferInternal function (used by several other contracts) also does not check the return value of ERC20 transfers.

18. The initialization method initManager() could be front-run, unless it is called within the same transaction as the creation of the contract.

The initialization method `initManager()` could be front-run, unless it is called within the same transaction as the creation of the contract.

Recommendation
Ideally the library API should be designed to prevent such mistakes.

19. AddressSet.sol

In `addAddressToSet(...)`, when `maintainList=True`, it does not check if there was an address already added with `maintainList=False` (when in `maintainList=False` a similar check is done). This will assign `index =1` to multiple addresses.

20. Possible misleading comment in Module.sol

In `Module.sol`, the following note is present:

```
/// Each module must implement the `init` method. It will be called when
/// the module is added to the given wallet.
```

There is, however, no abstract method `init()` defined and the only module implementing `init()` is the `LRCStakingModule.sol` module.

21. Consider removing the option to make delegateCalls from wallet and vault

Delegate calls allow for third party contracts to change the wallet storage variables, which include the owner, controller and the module's methods. They are not used by any of the modules.

22. Consider implementing a receive() function for no-data calls.

The wallet implements a fallback function that checks if the `msg.data` is different from "0x0". Solidity 0.6 onwards allows contracts to declare a `receive` function that is called only when no data is present, otherwise it goes through the `fallback`.

SOLIDIFIED

Audit Report for Loopring on May 21st, 2020.

Closing Summary

Loopring´s Hebao contracts contain sixteen issues, with three of them being critical, and five of major severity, along with several areas of note.

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

Disclaimer

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