SOLIDIFIED

Audit Report for Kirobo - June 22, 2021

Summary

Audit Report prepared by Solidified covering the Kirobo smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on June 22, 2021, and the results are presented here.

Audited Files

The source code has been supplied in a private source code repository:

https://github.com/kiroboio/ki-eth-contracts/ (branch: `develop`)

Commit number: `a21d9f8c8f3f45672da9e0237e29eca35e23c5ed`

UPDATE: Fixes received on October 3, 2021
Final commit number: `d32468e521760af8d75a87c651a6cc2adc2aa903`

Intended Behavior

Kirobo aims to protect users from incorrect cryptocurrency transfers. It works on two levels:

1. A transaction code that must be entered by the recipient in order to receive the transfer.
2. Funds retrieval capability allows the sender to retrieve the funds at any time, as long as the right code hasn’t been provided by the recipient.

Findings

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	Status	Comment
Code complexity	High	-
Code readability and clarity	Low	-
Level of Documentation	Low	-
Test Coverage	High	-

Issues Found

Solidified found that the Kirobo contracts contain no critical issues, 2 major issues, 4 minor issues, 2 warnings and 5 informational notes.

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

Issue #	Description	Severity	Status
1	RecoveryWallet.sol: TransferFrom does not work as intended	Major	Resolved
2	FactoryProxy.sol: Anyone can call setLocalEns() and set local ENS to an incorrect value	Major	Resolved
3	FactoryProxy.sol: Invalid casting	Minor	Resolved
4	Oracle.sol & RecoverOracle.sol: Locked ether	Minor	Resolved
5	Factory.sol: Function batchMultiSigCallPacked() allows address(0)	Minor	Resolved
6	RecoveryWallet.sol: Function sendEther() can potentially fail if ETH is being sent to a smart contract	Minor	Resolved
7	The contract can only be as safe as the input provided	Warning	-
8	Unaudited code	Warning	-
9	Proxy.sol: Use of reserved words in the smart contract	Note	Resolved
10	FactoryClaimable.sol and FactoryOwnable.sol: Functions' implementations are missing	Note	Resolved
11	RecoveryWallet.sol: functions transfer20() and transferFrom20() do not check the result of ERC20 transfer	Note	Resolved

12	Misc notes	Note	Resolved
13	Code clean-up	Note	Resolved

Critical Issues

No critical issues have been found.

Major Issues

1. `RecoveryWallet.sol`: TransferFrom does not work as intended

The methods `transferFrom721()`, `safeTransferFrom721()` and `safeTransferFrom721wdData()` do not make use of the `from` address passed as a parameter. This makes it transfer any asset from the current account rather than the intended `from` account.

The methods including `ERC20` transfer do not validate or return the result of the call.

Recommendation

Use the `from` address passed as the parameter to the transfer method.

Furthermore, return the result of `transfer` method in both `ERC20` and `ERC721` contracts to the caller.

Status

Resolved

2. `FactoryProxy.sol`: Anyone can call `setLocalEns()` and set local ENS to an incorrect value

This could either result in `FactoryProxy.sol` calling a different than expected target if ENS resolution is not checked or transactions would fail due to the unexpected ENS resolution value. (see usages of `_ensToAddress()` function in `FactoryProxy.sol`)

Recommendation

Consider adding the `multiSig2of3` modifier to the function.

Status

Resolved

Minor Issues

3. `FactoryProxy.sol`: Invalid casting

The methods `batchTransfer()` and `batchTransferPacked()` use `uint32` to calculate `beforeTS`, whereas the value expected is `uint40`. This can result in incorrect transaction expiry time and will prevent the transaction from execution.

Recommendation

Use `uint40` to retrieve the beforeTS value from the sessionId.

Status

Resolved

4. Oracle.sol & RecoverOracle.sol: Locked ether

The function `update20()` accepts ether but does not contain any method to withdraw the same from the contract. This can result in permanently locked ether.

Another such example can be found in `Wallet.executeBatchCall()`

Recommendation

Consider using a parameter that can be used as multi sig lock or return the `msg.value` after the function call.

Status

Resolved

5. Factory.sol: Function batchMultiSigCallPacked() allows address(0)

The function `batchMultiSigCallPacked()`, unlike it's unpacked counterpart method `batchMultiSigCall()`, allows `address(0)` as target address in `mcalls.mcall[j].to`.

Recommendation

Consider validating the input for `address(0)`.

Status

Resolved (function removed)

6. RecoveryWallet.sol: Function sendEther() can potentially fail if ETH is being sent to a smart contract

Function `sendEther()` calls `transfer()` when sending ETH to the `to` address, which only forwards 2300 gas. In cases where `to` is a smart contract whose fallback function consumes more than 2300 gas, the call will always fail. This will have the side effect of potentially preventing smart contracts (e.g. DAOs) from being sent any ETH.

For a more in depth discussion of issues with `transfer()` and smart contracts, please refer to: `https://diligence.consensys.net/blog/2019/09/stop-using-soliditys-transfer-now/`

Recommendation
Replace instances of `transfer()` and `send()` with `call()`.

Note
The same issue exists in functions `Trust.activateTrust()`, `Heritable.activateInheritance()` and `Backupable.activateBackup()`.

Status
Resolved

Warnings

7. The contract can only be as safe as the input provided

The contracts in OCW act as a giant proxy which redirects transactions based on the input. The functions use a set of input parameters to determine the transaction type, gas limit, data, value and almost all the properties related to a transaction. Furthermore, some methods accept input parameters encoded in a `bytes32` and use bit manipulation to decode the data.

This design can result in somewhat reduced gas usage, but is very prone to accidental invalid transactions and in some cases even loss of funds. One such example can be found in the test cases itself, where the documented parameters are not the one decoded by the method.

Moreover, the contracts accept any arbitrary target address and perform transactions, which also plays an important role in the safety of the smart contract.

Recommendation
Consider simplifying the contract a bit more to make it more readable and less error prone.

8. Unaudited code

The source repository does not contain the code for the interface `ICreator` and is not audited.

The wallet bytecode in the contract Wallet.sol and the related LLL codes were not audited.

Informational Notes

9. Proxy.sol: Use of reserved words in the smart contract

The function `staticcall()` and `call()` uses reserved names as the function names which is not recommended.

Recommendation

Avoid using reserved names as identifiers in the smart contract.

Status

Resolved

10. FactoryClaimable.sol and FactoryOwnable.sol: Functions' implementations are missing

Consider providing missing implementation.

Status

Resolved

11. RecoveryWallet.sol: functions transfer20() and transferFrom20() do not check the result of ERC20 transfer

Some ERC20 tokens' transfers do not revert, but return `false` instead. However, a successful transfer event will be emitted in such a case.

Recommendation

Consider using SafeERC20

Status

Resolved

12. Misc notes

Consider fixing the following items if it’s not intended.

1. Add `address(0)` validations throughout the contracts to prevent any accidental transfers. One such function where validation is required - `Factory.addWalletBackup()`. Status: Resolved.
2. `Wallet.sol`: Consider moving the msg.sender validation to the beginning of the method and outside the loop. Status: Resolved.

13. Code clean-up

1. `StorageBase.sol` - unused contract variable `s_debt`. Status: Resolved.
2. `Storage.sol` - unused contract variable `s_trust`. Status: Resolved.
3. `Factory.sol` - Line #432 - revert message misspelled `singer`. Status: Resolved.
4. `FactoryProxy.sol` - Revert message misspelled `gourp`. Status: Resolved.

5. `Interface.sol` - unused imports `./StorageBase.sol` and `../Trust.sol`. Status: Resolved.
6. `Trust.sol` - `Fund.cancelable` is set but never used. Status: Resolved.
7. `Backupable.sol` - provides another getter for `StorageBase.owner()` called `getOwner()`. Status: Resolved.
8. `Backupable.sol` - mixes usage of `s_owner` and `owner()` (they both mean the same). Status: Resolved.

Disclaimer

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