Ibis Code Overview

December 27, 2017

This document serves as a detailed overview of the design goals and decisions made in implementing the Ibis contracts. For a higher-level explanation of the purpose of the Ibis platform, please refer to the white paper and blog.

Overview

The Ibis contracts implement an ERC20 designed to facilitate the functionality specified in the white paper. In the expected case, these are limited to the routine operations of token exchange, transfers, and new charity approvals by the Ibis organization.

The majority of the code is intended to ensure robustness and security. In designing the system, we are particularly mindful of the following general threats:

- Key Theft: We design varying layers of defense-in-depth to mitigate damage by potential hacking up to and including total compromise of owner addresses.
- Economic Abuse: We implement a control mechanism to temporarily freeze undesirable behavior
- Insider Threat: Ibis users should not be required to entrust us with their money. As a result, we have taken steps to ensure that it impossible (assuming a majority of active and rational stake holders) for the Ibis organization itself to steal money.

The detailed implementaiton of these mitgation techniques are embedded in the module description sections comprising the remainder of this document.

Ibis

The Ibis contract contains the main business logic to run the system. It reads and writes critical user data from the standalone Core contract and inherits ownership and votable function modifiers from the Restricted and Democratic modules, respectively. The most commonly used functions in this module will be the deposit, transfer, and withdraw procedures.

The contract also holds upgrade logic. When a contract is upgraded, a predeclared *init* function of the new contract is invoked. The current contract self-destructs, sending the balance of ether to the new implementation. Except

for special circumstances, upgrades are invoked by multi-owner approval and must be approved by a simple majority stakeholder vote.

Finally, Ibis implements a somewhat involved mechanism for token freezing and redistribution. The most pressing economic threat to Ibis is the ability to trade Ibis tokens for other currencies via an authorized exchange as this would in a sense invalidate the purpose of Ibis as a charity token. While this threat cannot be entirely eliminated, this contract freezing mechanism allows the Ibis organization to have some power to mitigate abuse.

An Ibis organization owner address may instantly freeze any account at will. After a set amount of time has passed, frozen funds may be submitted for distribution amoung charities with majority stakeholder approval. The funds are then paritioned into more manageable sizes and distributed to random charity addresses.

The account freezing feature is primarily a regulatory tool. However, it also provides a convenient mechanism for fund recovery should an Ibis user lose his or her private account key. As a result, we can guarantee in the strictess sense that every piece of money that ever enters the Ibis ecosystem will indeed make its way to a charity.

Core

The Core module defines a stand-alone smart contract to house the critical application data needed to run the platform. This includes available user balances, the list of approved charities, and frozen user balances. The Core contract recognizes a single controller address at any time, which is the address of the contract that enforces the Ibis business logic. The controller address is able to edit a list of approved addresses that can modify the Core data. In the first iteration of Ibis, there are no extra approved addresses other than the main controller.

The data housed in the Core contract is intended to be persistent for the full lifetime of the Ibis ecosystem.

Restricted

Logic for regulating authoritative ownership of the contracts is housed in the Restricted module. The Restricted module recognizes a dynamic, homogenous set of contract owners as well as a single persistent *master address*.

Permissions for contract owners are fairly straight forward. More common operations such as adding and removing charity addresses may be done by a single owner. More sensitive operations such as changing system parameters or adding and removing owners must be approved by a certain threshold of the owner set.

The majority of owner addresses will be kept in cold storage to minimize the risk of compromise.

In the event that a malicious adversary gains access to owner keys, the worst damage can potentially be done right away. As a result, we also implement a *delay* functionality whereby sensitive operations can be forced to wait a certain number of blocks before they are executed. A majority owner vote can overrule such a delayed operation.

Finally, we also account for the worst-case scenario where all owner keys are somehow compromised. In the event that this happens, the master address, with the approval of the voting mechanism described later, has the ability to trigger a system *nuke*. In the event that this happens, all notions of ownership and charity statuses are revoked. Users of all types will be able to cash out their Ibis tokens for Ether.

The combination of these defense-in-depth mitigation measures ensures that we can recover from small degrees of corruption. Even in the event of total compromise, users will never have their tokens stolen or locked up forever.

Democratic

The measures implemented in the Restricted contract are sufficiently robust against external hacking. However, we would like to ensure that user do not have to trust the Ibis organization either.

Without voting abilities, the Ibis organization can trivially trigger a protocol upgrade that sends all ether to a new address of our choosing. To mitigate this abuse, the most signifcant operations (such as contract upgrades) must be subject to user voting.

To vote on an ongoing issue, users must temporarily lock up their tokens. In return, they will receive a proportional number of votes which they can apply toward supporting or rejecting any active votable issues. Users may withdraw their vote to retreive Ibis tokens at any time. When the voting period is closed, only the remaining active votes are counted to see whether a proposed operation should be executed.

Note that even frozen accounts are entitled to votes. This is a necessary measure to prevent the Ibis organization from sensoring dissenting users.

The democratic paradigm should instill a level of confidence in the fact that the Ibis organization cannot reasonably steal any user tokens. At the same time, votable issues can only be triggered by the contract owners, ensuring that the system cannot be corrupted by a tyranny of the majority. The evolution of Ibis will trend toward the status quo and any changes can only be realized through a robust system of checks and balances.

Conclusion

Security and trustworthess in the domain of charity is perhaps more important than any other. By carefully considering all attack vectors, we believe we have successfully designed an optimal balance in protecting the Ibis ecosystem. As always, if you have any questions, concerns, or suggestions, please feel free to raise them on any of our available forums. Thanks for reading!