NIIT Protocol (An interpretation of the Olympus Pro contract)

Design Architecture

The NIIt protocol was built on the Olympus pro protocol model. The protocol encourages users to hold the protocol tokens (NIIT) as opposed to selling, because there's a lot more profit to be gained from holding the tokens. These profits can be gained in two distinct ways: bonding of assets, or staking. The goal of the protocol is to ensure a rise in the market value of the protocol tokens, by ever increasing the demand for it.

The process of bonding involves a user, who holds an asset token acceptable by the protocol, bonding their assets in exchange for the protocol tokens at a discounted rate of 4%. This constraint to this is that the contract has to hold the asset tokens for a period of 7 days, before the equivalent worth of protocol tokens (NIIT) is released to the user. This ensures that at the end of the 7th day, the user has 4% more NIIT than he would have had, if he/she had just exchanged the assets directly for the tokens.

The staking process allows users to hold the protocol tokens (NIIT) much longer for a gain. The reward system used in the NIIT protocol is such that users are required to stake for a minimum of 7 days before the reward on the tokens staked can be calculated. The ROI is set at 50% per year, and calculated down to the second, thereby ensuring that a user's reward is aggregated every second. After the 7th day wait period, the accumulated stake plus reward is automatically re-compounded, after every action taken on the stake. The staking process can be done in two ways:

- staking directly from a mature bond
- staking the protocol token itself.

Design constants

Since the team was not provided access to any decentralised oracle that supports the Nahmi blockchain, we had to assume market value prices for the assets used in the project.

- 100 Nii === 1Niit
- 25 AssetTokens === 1Niit
- 4 Nii === 1 AssetTokens
- Price with bonds: 24 AssetToken === 1 NIIT (4% discount)

Contract Breakdown

The NIIt protocol includes 5 major contracts:

- BondDepository.sol
- Staking.sol
- NiitERC20.sol
- Vault.sol
- AssetERC20.sol

The ***BondDepository*** contract has 2 major functions:

1. *Deposit :* which takes in the amount of asset tokens the user wishes to bond, and the user's address as parameters. It creates a bond struct for the user, where it stores the user's information, and last time bonded, which the contract uses in calculating the bond's maturity.
2. *getTokens*: An external function called by the bond owner, to fund the user's wallet of the equivalent NIIT at bond maturity.

The ***Staking*** contract has 4 major functions:

1. *stakeFromMatureBonds:* An external function which checks the bond contract for existing/ mature bonds from the user, then stakes the proceeds of the bond directly for the user.
2. *stake*: An external function that takes the amount of protocol tokens to be staked, and stakes the tokens after it's transferred from the user's wallet.
3. *withdrawStake*: A function that allows the user, with a stake, withdraw his/her staked tokens plus accumulated rewards.
4. *_stake*: An internal function used by the ***stake*** and ***stakeFromMatureBonds*** functions. It is used to calculate and re-compound the stake reward based on the reward percentage and the last time staked.

The ***NiitERC20*** contract has 3 major functions:

1. *mintForSale:* An external function which allows users to mint the protocol tokens using the blockchain native coin (NII) at a market ratio of 100 NII tokens to NIIT.
2. *mintFromBond*: An external function, called through the bondDepository contract to mint protocol tokens for users upon bond maturity.
3. mintFromStake: An external function, called through the staking contract to mint protocol tokens for stakeholders.

Meet The Team

- **Abdulrasheed Adediran (Software Developer)**
  Contribution: Frontend architecture, design and markup build

- **Boladale-Lawal Olajumoke (Software Developer)**
  Contribution: Smart contract architecture, system design, writing and testing smart contract, frontend--smart contract integrations

- **Emmanuel Enebeli (Software Developer)**
  Contribution: Frontend-smart contract Integrations.