HedgeyOTC Technical Documentation

Contract: HedgeyOTC.sol

Background: Hedgey Finance is the group that develops token tools for DAOs to utilize their treasury tokens along various life cycle events of the DAO, from inception to maturity. A DAO, generally known as Decentralized Autonomous Organization, will be used generally as a term to describe a group of people in the "Web3" ecosystem, namely building products in the Ethereum and similar EVMs Blockchains. DAOs often have a Token, generally an ERC20 standardized Token, that represents some form of governance, purpose, or just general community fun that is associated with the DAO itself. DAOs use these tokens in various ways, this specific protocol is aimed to provide a means for DAOs and large holders to sell tokens over the counter (OTC) to possible buyers. Over the counter transactions have long happened with large holders and the DAO itself, but generally speaking have been done through back door channels over telegram or discord, and often fraught with scam and fraud. This contract solves this trust issue by making an escrowless, trustless contract that both parties can feel confident using whereby there is no need to trust an individual intermediary. Additionally, the contract has the benefit of building on top of the Hedgeys FuturesNFT contract, such that the DAO may decide to sell locked tokens, rather than immediately liquid tokens. Selling locked tokens helps regulate the timing when tokens leave the DAO treasury and enter liquid markets and the true circulating supply.

Functional Overview

Purpose: The purpose of the HedgeyOTC contract is to make a very simple and trustless experience for DAOs and large token holders to sell unlocked or locked tokens to buyers – either a singly Buyer or public buyers in general. When the tokens are locked, they leverage the power of the FuturesNFT contract to lock and mint NFTs to each buyer. OTC sales are important for DAOs and large holders because often in liquid token markets there is not sufficient liquidity to responsibly sell those tokens without large and unwanted price impacts.

Roles: There are two roles in the HedgeyOTC protocol, the Seller and the Buyer:

a. **Seller:** The Seller is the DAO, or token holder generally who is looking to sell some amount of their tokens over the counter. They may want to do this directly with a known buyer, in which case they can define that exact buyer to be only allowed to purchase, or they may want to sell tokens to the general public. The Seller is responsible for defining all of the parameters of the sale, ie the amount of tokens they want to sell, the currency they want to receive in exchange (payment currency), who they are selling to, how long the sale is available for, the minimum someone can buy (if public), and whether they should be locked or not. The Seller will start the sale by inputting all the info about it, and then their tokens will be pulled into the contract to be held for buyers to buy. The Seller may also cancel their open sale at anytime should they wish.

b. **Buyer**: The Buyer is the individual or DAO who is interested in purchasing tokens from the seller. The Buyer may have agreed to a preset price with the seller, and so is a private deal, or the buyer may be an individual who discovers the sale and identifies the price as acceptable and so is interested in purchasing the tokens. The buyer simply needs to know the numeric id of the sale and how many tokens the buyer would like to purchase. The buyer will then buy the amount of tokens, paying their pro-rata amount for the tokens, and either receive unlocked tokens or a Hedgey NFT that represents the tokens the purchased locked in the FuturesNFT contract. They can then later unlock those tokens interacting directly with the FuturesNFT.sol contract.

Functions: There are three writable functions in the contract, which involves the setup of a sale, closing that sale, and buying some tokens from the ongoing sale.

1. **create**
   The create function is called by the Seller, which sets up the Sale. The Sale has the parameters defined by the Seller of the token they want to sell, the amount of tokens to be sold, what token/currency they want to receive in exchange for the sale, the price they are selling the tokens, how long the sale is going to be open for buyers to participate, a minimum allowable amount of tokens that can be purchased, an optional unique address for a specific buyer, and an unlockdate option for tokens that are going to be sold as locked tokens. The create function will pull the tokens the seller is selling into the contract to be held so that buyers can purchase them. The Sale is considered active after the function is called, while there is still a remaining balance of tokens to be sold, while it has not been closed, and before the maturity date. After all tokens have been sold, or the maturity date has passed, or the Seller has manually closed the sale, the Sale is no longer active.
2. **close**
   The close function can only be called by a Seller of an existing Sale, when they want to cancel or close the ongoing sale. This is a critical function for if a mistake on price or amount is made, or if the Seller simply doesn't want to sell any more tokens, or if the maturity of the sale has passed and is thus no longer active. This function will return any remaining tokens that haven't been sold back to the Seller, and close the ongoing sale.
3. **buy**
   The buy function is used by the Buyers to purchase an amount of tokens from an ongoing sale. The buyers will define the sale they want to participate in, and how many tokens they want to buy. If the Sale is explicitly whitelisted for a single buyer, then only that buyer may participate. Additionally, the buyers are subject to purchasing a minimum number of tokens defined by the Seller in the initial create stage. Buyers may receive either unlocked tokens or an NFT of locked tokens, depending on the Sale. Buyers will pay the purchase amount, equal to the price of the tokens times the amount of tokens to be bought, and that purchase amount will be delivered directly to the seller. Buyers may only buy the remaining amount of tokens that are still for Sale – this prevents attempted purchases from failing in the case that there is not sufficient tokens in the contract left to be purchased. There is no limit on the number of times a buyer may participate, they can purchase tokens in multiple transactions should they choose – so long as the Sale is active.

Technical Overview

Contract Dependencies and Imports

1. OpenZeppelin **ReentrancyGuard.sol**
   Purpose: Prevent reentrancy attacks for contract function calls that transfer tokens and change storage

2. Interface **Decmials.sol**
   Purpose: Used in the pricing calculation for the purchase price of the buyers. The formula of amount * price needs to be divided by token decimals. This is used to get the decimals of the token so that the price can be calculated accurately.

3. Library **TransferHelper.sol**
   Purpose: Assist with transferring tokens between users and contracts. This is a helper library that ensures the correct amount of tokens is transferred and that the before and after balances match with what is the expected outcome. Additionally the library checks to ensure sufficient balances are owned by the transferrer prior to attempting to transfer the tokens.

4. Library **NFTHelper.sol**
   Purpose: Assist with locking tokens into a FuturesNFT contract, which mints an NFT and locks tokens with a precise unlock date. The NFT acts as the key for the owner to unlock the tokens after the unlockDate has been passed.

Global Variables

1. *address payable public* **weth**
   This is the address of the contract 'WETH' or Wrapped Ether, which assists with storing Ether in this contract as an ERC20. It was chosen to use WETH to store in the contract to make moving tokens back and forth consistent – as well as ensuring that the proper amount of ETH is received or delivered when buying or selling.

2. *uint public* **d**
   d is a basic counter, that is used to identify each Deal in storage. It is incremented each time a new Deal (sale) is create.

3. *address public* **futureContract**
   This is the address of the Hedgeys NFT (FuturesNFT.sol) contract that is used for locking the tokens. It is specified as a global variable so that this contract can only lock tokens with this FuturesNFT contract, ensuring safety and security for buyers who are purchasing locked tokens.

4. *struct* **Deal**
   Deal is the main object that stores all of the parameters and details of each Deal or Sale. It has the following data params which collectively define an active Deal:
   address seller: this is the Seller, who is the msg.sender when the create function is called

   address token: the ERC20 contract address of the token being sold, delivered by the Seller

   address paymentCurrency: the ERC20 contract address of the token that is used to purchase tokens being sold – delivered by the buyer

   uint256 remainingAmount: this is the current amount of tokens that remain to be sold in the active deal. At initial creation, the remainingAmount is set to the total amount, then as each buyer participates this amount is reduced until it reaches 0. Once at 0 the Deal is deleted and is no longer active. Denominated in the Token currency

   uint256 minimumPurchase: This represents the minimum amount of tokens that any single buyer may purchase, preventing unwanted miniscule amounts of purchases to occur. This must be greater than 0 and cannot exceed the total amount set initially. Denominated in the Token currency

   uint256 price: this is the price per token that buyers must pay to purchase the tokens being sold. This is denominated in the payment currency ie think of a sale of 100 tokens of XYZ being sold at a price of 5 USDC per 1 XYZ as a price of 5.

   uint256 maturity: this is a Unix timestamp date (calculated against the block.timestamp) that is used to signify when a deal is no longer active. This is helpful for ensuring that sales to not become stale and forgotten with unwanted consequences.

   uint256 unlockDate: If the tokens are to be locked in an NFT for the future unlock, this value represents the unlock timestamp in Unix time. This value may also be 0.

   address buyer: This address is a specific address of a buyer for private deals. If the address is the 0 address, then there is no specific buyer, and anyone can participate.

5. *mapping(uint256 => Deal) public* **deals**
   This mapping maps the uint d to each Deal struct in storage. This is a simple mechanic for identifying in storage the location of each Deal struct that can be accessed for buying, closing, and viewing the deals.

Write Functions

1. *function* **create**(*address* _token, *address* _paymentCurrency, *uint256* _amount, *uint256* _min, *uint256* _price, *uint256* _maturity, *uint256* _unlockDate, *address payable* _buyer) *external payable nonReentrant*
   The Create function is used by a seller to create a new active Deal. The function takes several parameters that define the deal terms, which are then stored in a Deal struct based on the current incrementer d. 'd' is incremented during this process as well. Tokens are pulled into the contract from the Seller and then an event highlighting all of the details is emitted. The msg.sender is established as the Seller of the Deal. There are a few checks to make sure that the details of the deal will make a successful active deal, such as checking that the maturity date is in the future, checking that the amount is greater than the min, and checking that minimum purchase amount (defined as the min * price / tokenDecimals) will not result in a 0 from integer rounding.
   address _token: the ERC20 contract address of the token being sold, this token will get pulled into the contract upon calling the function

   address _paymentCurrency: the ERC20 contract address of the token that is used to make purchases – this is what the seller will get paid in (typically WETH address or USDC or DAI)

   uint256 _amount: this is the total amount of tokens that will be sold during this deal. This is stored in the remainingAmount struct data when first created.

   uint256 minimumPurchase: This represents the minimum amount of tokens that any single buyer may purchase, preventing unwanted miniscule amounts of purchases to occur. This must be greater than 0 and cannot exceed the total amount set initially. Denominated in the Token currency

   uint256 price: this is the price per token that buyers must pay to purchase the tokens being sold. This is denominated in the payment currency ie think of a sale of 100 tokens of XYZ being sold at a price of 5 USDC per 1 XYZ as a price of 5.

   uint256 maturity: this is a Unix timestamp date (calculated against the block.timestamp) that is used to signify when a deal is no longer active. This is helpful for ensuring that sales to not become stale and forgotten with unwanted consequences.

   uint256 unlockDate: If the tokens are to be locked in an NFT for the future unlock, this value represents the unlock timestamp in Unix time. This value may also be 0.

   address buyer: This address is a specific address of a buyer for private deals. If the

address is the 0 address, then there is no specific buyer, and anyone can participate.

2. *function* **close**(*uint256 _d*) *external nonReentrant*
   This function is used by the Seller to close a deal that is active or inactive, but has tokens remaining to be purchased (in the case where the maturity date has passed). The function will pull the Deal struct from memory based on the _d input, check that the msg.sender is in fact the Seller, and check that the remaining amount is still greater than 0 (ie still active). If those two are true, then the function will return the remainingAmount of tokens in the Deal, and delete the Deal from storage so that it can no longer be accessed.

3. *function* **buy**(*uint256 _d, uint256 _amount*) *external payable nonReentrant*
   This is the only function used by the Buyers to purchase tokens from an active Sale. The input is simple the id of the Deal struct that is stored in storage, and the amount of tokens they are purchasing. For typical Hedgey UI, we make the id easy by showing all of the sale details so that when users click on the sale, they are intentionally interacting and purchasing tokens from the intended sale. The function will lookup the deal in storage by the id, keep in memory and first complete a few basic checks. It will make sure that the deal is active and the maturity date is in the future, and will check that the msg.sender can actually participate – ie if the deal has a single buyer listed, then the msg.sender must be that buyer; if the deal has the 0 address as the buyer than anyone can participate. Finally it will check that the amount being purchased is within logic realm, that it is greater than the minimum or equal to the entire remaining amount, and that there is sufficient remaining amount to cover the purchase; remaining amount greater than the amount to be bought. Having passed this, the function will calculate the total purchase price, which is effectively the price * amount, factored by the token decimals, and then transfer the paymentCurrency from the buyer (msg.sender) to the seller. It will next check whether the tokens should be locked or unlocked based on the unlock date, and if they are to be locked, deliver the tokens to the FuturesNFT contract and mint an NFT to the buyer representing their locked tokens; otherwise it will simply deliver the unlocked tokens to the buyer. Finally, the function reduces the remaining amount of tokens in the storage Deal struct – and if that now is 0, it will delete the variable and close the deal, making it inactive.