Hedgey Token Lockup and Vesting Plans

Date	June 2023
Auditors	David Braun, Chingiz Mardanov

1 Executive Summary

This report presents the results of our engagement with **Hedgey** to review **Token Lockup and Vesting Plans**.

The review was conducted over four weeks, from **June 26, 2023** to **July 21, 2023**, by **Chingiz Mardanov** and **David Braun**. A total of 29 person-days were spent.

2 Scope

Our review focused on the commit hash `6a5ff58c2e83015b83c8de15f1cc61e9ac58f2c7`. The list of files in scope can be found in the Appendix.

2.1 Objectives

Together with the **Hedgey** team, we identified the following priorities for our review:

1. Correctness of the implementation, consistent with the intended functionality and without unintended edge cases.
2. Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Best Practices, and the Smart Contract Weakness Classification Registry.
3. Reentrancy Attacks
4. Errors with segmenting and combining lockup plans that could cause a user to lose out on funds, or receive additional funds they were not meant to get
5. Errors with segmenting and combining lockup plans that allow someone's to unlock their tokens earlier than scheduled initially
6. Errors with On-chain voting vaults that would allow someone to pull their tokens from the contract without authorization
7. Calculation errors from the time lock library

2.2 Update: August 3rd - Mitigations

The report was updated to reflect mitigations implemented for the findings. An additional 10 person days (in the week of July 31 - August 4) were spent to conduct the review, focusing on reviewing the changes that were implemented addressing the specific findings. We have also included the `PlanDelegator.sol` contract into the scope of the review.

3 Security Specification

This section describes, **from a security perspective**, the expected behavior of the system under audit. It is not a substitute for documentation. The purpose of this section is to identify specific security properties that were validated by the audit team.

3.1 Actors

The relevant actors are listed below with their respective abilities:

- **Vesting or a Lockup Plan Holder** - also the recipient of the vested tokens. Wallet or contract that is the owner of a specific ERC-721 token.
- **Vesting Admin** - address that is in certain cases capable of moving the ERC-721 plan on behalf of the holder as well as revoke the plan.
- **Vesting Plan** - a linear token unlock plan that is revokable by the vesting admin. Can also be only transferred by vesting admin when that was enabled on creation.
- **Voting Vesting Plan** - same as vesting plan but with additional Voting Vaults to support on-chain governance.
- **Lockup Plan** - linear token unlock plan, that can be transferred, segmented and combined again.
- **Voting Lockup Plan** - same as lockup plan but with additional of Voting Vaults to support on-chain governance.
- **Soul Bound Lockup Plan** - same as lockup plan but without an ability to transfer it.

3.2 Trust Model

We are delighted to highlight the inherent decentralized nature of the Hedgey platform. Upon conducting a comprehensive review of the contracts, we find it commendable that no upgradeability functionality is incorporated, a decision which aligns well with our principles at Diligence.

However, it is crucial to address one centralization risk that warrants attention. This concern pertains to the vesting plans that allow the `adminTransferOBO` flag to be enabled. In such instances, the vesting plan's administrator gains the ability to transfer tokens on behalf of the recipient, potentially leading to the loss of vested but unclaimed tokens. While we understand that this measure is intended to safeguard less experienced users from the risk of losing access to their vesting wallets, it also introduces the possibility of malicious activities.

4 Findings

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

4.1 Lockup Plans Are Not Well Suited for Trading on Traditional OTC Platforms Medium

Description

For most of the OTC trading platforms with RFQ style the maker or the taker creates an order that is valid for some time and is expecting a specific token ID. In case of a lockup period a trade participants can request to buy a specific plan ID and then give a fixed amount of time to fill that order, assuming that anything past that time that is unvested is guaranteed to go to them. In reality, the taker of such an order can batch two transactions in one block:

1. Segment the `planId` the order is expecting into two: one with just 1 wei to vest and the other with the rest. The large plan will have an incremented plan ID. The small plan will have the old ID.
2. Fill the order and get the full payment for what is now a worthless plan token.

People should be aware of such a possibility before attempting to purchase any lockup plans over OTC platforms.

Recommendation

One way to solve this is to assign both plans a new ID during the segmentation process.

4.2 Architectural Pattern of Internal and External Functions Increases Attack Surface Minor ✓ Fixed

Resolution

Fixed as of commit `f4299cdba5e863c9ca2d69a3a7dd554ac34af292`.

Description

There is an architectural pattern throughout the code of functions being defined in two places: an external wrapper (`name`) that verifies authorization and validates parameters, and an internal function (`_name`) that contains the implementation logic. This pattern separates concerns and avoids redundancy in the case that more than one external function reuses the same internal logic.

For example, `VotingTokenLockupPlans.setupVoting` calls an internal function `_setupVoting` and sets the `holder` parameter to `msg.sender`.

contracts/LockupPlans/VotingTokenLockupPlans.sol:L164-L165

```
function setupVoting(uint256 planId) external nonReentrant returns (address votingVault) {
  votingVault = _setupVoting(msg.sender, planId);
```

contracts/LockupPlans/VotingTokenLockupPlans.sol:L436-L437

```
function _setupVoting(address holder, uint256 planId) internal returns (address) {
  require(ownerOf(planId) == holder, '!owner');
```

In this case, however, there is no case in which `holder` should not be set to `msg.sender`. Because the internal function doesn't enforce this, it's theoretically possible that if another internal (or derived) function were compromised then it could call `_setupVoting` with `holder` set to `ownerOf(planId)`, even if `msg.sender` isn't the owner. This increases the attack surface through providing unneeded flexibility.

Other Examples

contracts/LockupPlans/TokenLockupPlans.sol:L107-L113

```
function segmentPlan(
  uint256 planId,
  uint256[] memory segmentAmounts
) external nonReentrant returns (uint256[] memory newPlanIds) {
  newPlanIds = new uint256[](segmentAmounts.length);
  for (uint256 i; i < segmentAmounts.length; i++) {
    uint256 newPlanId = _segmentPlan(msg.sender, planId, segmentAmounts[i]);
```

contracts/LockupPlans/TokenLockupPlans.sol:L244-L245

```
function _segmentPlan(address holder, uint256 planId, uint256 segmentAmount) internal returns (uint256 newPlanId) {
  require(ownerOf(planId) == holder, '!owner');
```

contracts/VestingPlans/TokenVestingPlans.sol:L115-L117

```
function revokePlans(uint256[] memory planIds) external nonReentrant {
  for (uint256 i; i < planIds.length; i++) {
    _revokePlan(msg.sender, planIds[i]);
```

contracts/VestingPlans/TokenVestingPlans.sol:L226-L228

```
function _revokePlan(address vestingAdmin, uint256 planId) internal {
  Plan memory plan = plans[planId];
  require(vestingAdmin == plan.vestingAdmin, '!vestingAdmin');
```

Recommendation

To reduce the attack surface, consider hard coding parameters such as `holder` to `msg.sender` in internal functions when extra flexibility isn't needed.

4.3 Vesting Admin Could Prevent the Recipient From Redeeming Minor ✓ Fixed

Resolution

Fixed as of commit `f4299cdba5e863c9ca2d69a3a7dd554ac34af292` by adding new function `toggleAdminTransferOBO` to contracts `TokenVestingPlans` and `VotingTokenVestingPlans`.

Description

In the vesting part of the protocol, each plan has a vesting admin who can transfer tokens on behalf of the plan holder. However, this setup poses a risk of centralization. For instance, a plan holder might leave their tokens vested for a long time without claiming them. Then, if the vesting admin decides to transfer the plan to a different wallet, the recipient may never be able to claim those tokens.

We understand that this feature is meant to assist novice users who might lose their private keys and need a safety net. Nevertheless, we suggest giving the plan recipient the option to toggle the `adminTransferOBO` on and off. This way, they can protect themselves better against any potentially malicious actions from the vesting admin, all without triggering a taxable event.

4.4 Revoking Vesting Will Trigger a Taxable Event Minor ✓ Fixed

Resolution

Fixed as of commit `f4299cdba5e863c9ca2d69a3a7dd554ac34af292`.

Description

From the previous conversations with the Hedgey team, we identified that users should be in control of when taxable events happen. For that reason, one could redeem a plan in the past. Unfortunately, the recipient of the vesting plan can not always be in control of the redemption process. If for one reason or another the administrator of the vesting plan decides to revoke it, any vested funds will be sent to the vesting plan holder, triggering the taxable event and burning the NFT.

Examples

contracts/VestingPlans/TokenVestingPlans.sol:L226-L237

```
function _revokePlan(address vestingAdmin, uint256 planId) internal {
  Plan memory plan = plans[planId];
  require(vestingAdmin == plan.vestingAdmin, '!vestingAdmin');
  (uint256 balance, uint256 remainder, ) = planBalanceOf(planId, block.timestamp, block.timestamp);
  require(remainder > 0, '!Remainder');
  address holder = ownerOf(planId);
  delete plans[planId];
  _burn(planId);
  TransferHelper.withdrawTokens(plan.token, vestingAdmin, remainder);
  TransferHelper.withdrawTokens(plan.token, holder, balance);
  emit PlanRevoked(planId, balance, remainder);
}
```

contracts/VestingPlans/VotingTokenVestingPlans.sol:L245-L263

```
function _revokePlan(address vestingAdmin, uint256 planId) internal {
  Plan memory plan = plans[planId];
  require(vestingAdmin == plan.vestingAdmin, '!vestingAdmin');
  (uint256 balance, uint256 remainder, ) = planBalanceOf(planId, block.timestamp, block.timestamp);
  require(remainder > 0, '!Remainder');
  address holder = ownerOf(planId);
  delete plans[planId];
  _burn(planId);
  address vault = votingVaults[planId];
  if (vault == address(0)) {
    TransferHelper.withdrawTokens(plan.token, vestingAdmin, remainder);
    TransferHelper.withdrawTokens(plan.token, holder, balance);
  } else {
    delete votingVaults[planId];
    VotingVault(vault).withdrawTokens(vestingAdmin, remainder);
    VotingVault(vault).withdrawTokens(holder, balance);
  }
  emit PlanRevoked(planId, balance, remainder);
}
```

Recommendation

One potential workaround is to only withdraw the unvested portion to the vesting admin while keeping the vested part in the contract. That being said `amount` and `rate` variables would need to be updated in order not to allow any additional vesting for the given plan. This way plan holders will not be entitled to more funds but will be able to redeem them at the time they choose.

4.5 Use of `selfdestruct` Deprecated in `VotingVault` Minor ✓ Fixed

Resolution
Fixed as of commit `f4299cdba5e863c9ca2d69a3a7dd554ac34af292`.

Description

The `VotingVault.withdrawTokens` function invokes the `selfdestruct` operation when the vault is empty so that it can't be used again.

The use of `selfdestruct` has been deprecated and a breaking change in its future behavior is expected.

Examples

contracts/sharedContracts/VotingVault.sol:L36-L39

```
function withdrawTokens(address to, uint256 amount) external onlyController {
    TransferHelper.withdrawTokens(token, to, amount);
    if (IERC20(token).balanceOf(address(this)) == 0) selfdestruct;
}
```

Recommendation

Remove the line that invokes `selfdestruct` and consider changing internal state so that future calls to `delegateTokens` always revert.

4.6 Balance of `msg.sender` Is Used Instead of the `from` Address Minor ✓ Fixed

Resolution
Fixed as of commit `f4299cdba5e863c9ca2d69a3a7dd554ac34af292`.

Description

The `TransferHelper` library has methods that allow transferring tokens directly or on behalf of a different wallet that previously approved the transfer. Those functions also check the sender balance before conducting the transfer. In the second case, where the transfer happens on behalf of someone the code is checking not the actual token spender balance, but the `msg.sender` balance instead.

Examples

contracts/libraries/TransferHelper.sol:L18-L25

```
function transferTokens(
  address token,
  address from,
  address to,
  uint256 amount
) internal {
  uint256 priorBalance = IERC20(token).balanceOf(address(to));
  require(IERC20(token).balanceOf(msg.sender) >= amount, 'THL01');
```

Recommendation

Use the `from` parameter instead of `msg.sender`.

4.7 Refactor Large Functions for Readability

Description

Function bodies larger than a typical screenful of text are harder to read and to reason about security properties.

Examples

- `VotingTokenLockupPlans._combinePlans` is 98 lines long.
- `VotingTokenLockupPlans._segmentPlan` is 72 lines long.
- `TokenLockupPlans._segmentPlan` is 66 lines long.

Recommendation

Refactor large functions into compositions of smaller, easier-to-understand functions.

4.8 Unused Code in Source Files ✓ Fixed

Resolution
Fixed as of commit `f4299cdba5e863c9ca2d69a3a7dd554ac34af292`.

Description

There is unused code in the source files.

Examples

The function `TimelockLibrary.totalPeriods` isn't used and appears to be incorrect (rounds down instead of up).

contracts/libraries/TimelockLibrary.sol:L28-L31

```
/// @notice function to calculate the total periods in a given plan based on the rate and amount
function totalPeriods(uint256 rate, uint256 amount) internal pure returns (uint256 periods) {
  periods = amount / rate;
}
```

`ERC721Delegate.wasTransferred` is written but not read:

contracts/ERC721Delegate/ERC721Delegate.sol:L26-L27

```
// Mapping if a token has been transferred
mapping(uint256 => bool) public wasTransferred;
```

Recommendation

Remove unused code to reduce confusion and to decrease the attack surface.

4.9 Use Custom Errors to Save Gas

Description

As of Solidity 0.8.4 it is possible to save gas when reporting error conditions by using custom errors instead of strings.

Examples

contracts/ERC721Delegate/ERC721Delegate.sol:L40

```
require(index < ERC721.balanceOf(owner), 'ERC721Enumerable: owner index out of bounds');
```

contracts/ERC721Delegate/ERC721Delegate.sol:L59

```
require(index < totalSupply(), 'ERC721Enumerable: global index out of bounds');
```

Recommendation

We recommend using custom errors to save gas.

4.10 Use `_beforeTokenTransfer` to Override Behavior in OpenZeppelin Token Contracts

Partially Addressed

Resolution
As of commit `f4299cdba5e863c9ca2d69a3a7dd554ac34af292`, `TokenLockupPlans_Bound` and `VotingTokenLockupPlans_Bound` now use `_beforeTokenTransfer` but `TokenVestingPlans` and `VotingTokenVestingPlans` still do not. Client response: > Did not implement for the Vesting plans because the impact would override and complicate functionality desired through the vestingAdminTransferOBO, because of the way the hooks process before and after it would require significant and possibly risky changes

Description

Contracts such as `TokenVestingPlans`, `VotingTokenVestingPlans`, `TokenLockupPlans_Bound`, and `VotingTokenLockupPlans_Bound` add special conditions for allowing the transfer of tokens by overriding the `transferFrom`, `_safeTransfer`, and `_transfer` functions in OpenZeppelin token contracts. While workable this approach can be error-prone and may break during future upgrades to the underlying contracts.

For example, in the unreleased version of OpenZeppelin's contracts, the `ERC20._transfer` function is no longer virtual and contains the warning:

> NOTE: This function is not virtual, {_update} should be overridden instead.

Examples

contracts/VestingPlans/TokenVestingPlans.sol:L282

```
function transferFrom(address from, address to, uint256 tokenId) public override(IERC721, ERC721) {
```

contracts/VestingPlans/TokenVestingPlans.sol:L291

```
function _safeTransfer(address from, address to, uint256 tokenId, bytes memory data) internal override {
```

contracts/LockupPlans/NonTransferable/TokenLockupPlans_Bound.sol:L21

```
function _transfer(address from, address to, uint256 tokenId) internal virtual override {
```

Recommendation

OpenZeppelin recognizes this as a common use case and provides a hook for cleaner control over transfer behavior. Use the _beforeTokenTransfer hook with version 4 contracts to enforce transfer conditions.

Please note however that the `_beforeTokenTransfer` hook will be deprecated in the next release of OpenZeppelin's contracts in favor of a new function called `_update`.

4.11 Use `calldata` Instead of `memory` for External Function Arguments Data Location Partially Addressed

Resolution
Fixed for some functions but not others, e.g., `TokenLockupPlans.segmentPlan`, `TokenLockupPlans.segmentAndDelegatePlans`, `VotingTokenLockupPlans.segmentPlan`, and `VotingTokenLockupPlans.segmentAndDelegatePlans`.

Description

Reference types (e.g., arrays, mappings, and structs) in function arguments must declare the "data location" for where they are stored. There are two options for external functions: `calldata` or `memory`. `calldata` arguments are immutable which reduces complexity and improves code readability. `memory` arguments are mutable and add an implicit copy operation.

Examples

contracts/LockupPlans/TokenLockupPlans.sol:L72

```
function redeemPlans(uint256[] memory planIds) external nonReentrant {
```

contracts/VestingPlans/VotingTokenVestingPlans.sol:L123

```
function revokePlans(uint256[] memory planIds) external nonReentrant {
```

contracts/LockupPlans/TokenLockupPlans.sol:L107-L110

```
function segmentPlan(
  uint256 planId,
  uint256[] memory segmentAmounts
) external nonReentrant returns (uint256[] memory newPlanIds) {
```

Recommendation

The Solidity documentation makes the following recommendation:

> If you can, try to use `calldata` as data location because it will avoid copies and also makes sure that the data cannot be modified.

We recommend always using `calldata` for external function parameters unless doing so would incur a serious performance penalty or make code harder to read.

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
contracts/ERC721Delegate/ERC721Delegate.sol	`4c6d778a225ff249d285341be3a128a24430260a`

File	SHA-1 hash
contracts/LockupPlans/NonTransferable/TokenLockupPlans_Bound.sol	529f8422ca5c1b8312df594a2ebc93063b08e0bc
contracts/LockupPlans/NonTransferable/VotingTokenLockupPlans_Bound.sol	fce83a3ec6e677ca92a445411f1d2c0b2a88540c
contracts/LockupPlans/TokenLockupPlans.sol	3ab057c1df70042c6b4ee65c261cbffa3784c295
contracts/LockupPlans/VotingTokenLockupPlans.sol	98d06ee151c87e456d2dd63c1bac766369cf1487
contracts/Periphery/BatchPlanner.sol	c0d3c73b59371afc5ee1312156105b2ff778385f
contracts/Periphery/ClaimCampaigns.sol	43d0d3d734e398c2a4f184bc362548d4bbbb4920
contracts/VestingPlans/TokenVestingPlans.sol	ca8c0e8934d2aff4130edf599b347a35ace5431e
contracts/VestingPlans/VotingTokenVestingPlans.sol	1f8fe33624358e9787cef581bdb46c797f19c333
contracts/interfaces/IDelegateNFT.sol	26b381ac7b2a987f261682c4ebb989efbee06f97
contracts/interfaces/ILockupPlans.sol	4393120bef23f18e900ebe3f22a475bfa2d59ff9
contracts/interfaces/IVestingPlans.sol	74862d6fe439c6f85582a6726b787a80341e1d2e
contracts/libraries/TimelockLibrary.sol	b9d052a25ebaa233056bd9fbd4523781cbede99c
contracts/libraries/TransferHelper.sol	a64d729331d35d311a1b62da06f3c970cb508194
contracts/sharedContracts/LockupStorage.sol	51360e24db40eaa733012d54e3bfa0a5023455e9
contracts/sharedContracts/URIAdmin.sol	a358e8ff9f73137a9cc1d86219b7aa5dff9bee00
contracts/sharedContracts/VestingStorage.sol	520dd9cb3a534bf81790ff4d3ad6bfdf8c95ab9f
contracts/sharedContracts/VotingVault.sol	3c424708fee57ef12e55e2e6ccdd8ab4bc8636a8

Appendix 2 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of code and only the code we note as being within the scope of our review within this report. Any Solidity code itself presents unique and unquantifiable risks as the Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond specified code that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. In some instances, we may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.