SALUS SECURITY

MAY 2024

CODE SECURITY ASSESSMENT

HEDGEY

Overview

Project Summary

- Name: Hedgey - Locked vesting tokenPlans
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/hedgey-finance/Locked_VestingTokenPlans
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Hedgey - Locked vesting tokenPlans
Version	v1
Type	Solidity
Dates	May 17 2024
Logs	May 17 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	3
Total informational issues	1
Total	4

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	The campaign's manager can withdraw funds at any time	Low	Centralization	Pending
2	Malicious users can temporarily cause a DoS in the protocol	Low	Business Logic	Pending
3	The token may be locked in the vault after the vault has been deleted	Low	Business Logic	Pending
4	Missing events for functions that change critical state	Informational	Logging	Pending

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. The campaign's manager can withdraw funds at any time	
Severity: Low	Category: Centralization
Target: - contracts/Periphery/ClaimCampaigns.sol	

Description

In the Hedgey protocol, each token release event has a manager role, which allows the manager to withdraw funds at any stage of the token release process. This means that even after users receive proof of token release, the manager still retains the right to withdraw those funds.

contracts/Periphery/ClaimCampaigns.sol

```
function cancelCampaign(bytes16 campaignId) external nonReentrant {
    Campaign memory campaign = campaigns[campaignId];
    require(campaign.manager == msg.sender, '!manager');
    delete campaigns[campaignId];
    delete claimLockups[campaignId];
    TransferHelper.withdrawTokens(campaign.token, msg.sender, campaign.amount);
    emit CampaignCancelled(campaignId);
}
```

Such a centralized design undermines the credibility of the Hedgey protocol. This design immediately affects users' perception of trustworthiness, potentially leading them to exit the protocol and causing a reduction in the number of protocol users.

Recommendation

Consider adding a lock-up period. Only the manager can withdraw all funds after the lock-up period.

2. Malicious users can temporarily cause a DoS in the protocol

Severity: Low	Category: Business Logic
Target: - contracts/Periphery/ClaimCampaigns.sol	

Description

When creating a campaign in the Hedgey protocol, users can customize the ID.

contracts/Periphery/ClaimCampaigns.sol:L124-L156

```
function createUnlockedCampaign(
    bytes16 id,
    Campaign memory campaign,
    Donation memory donation
) external nonReentrant {
    require(!usedIds[id], 'in use');
    usedIds[id] = true;
    ...
}
```

In the highlighted code above, duplicate IDs will cause transactions to revert. This means that malicious users can cause all transactions creating campaigns to revert by front-run attack. While this attack requires consuming some gas, malicious actors can still put the protocol into a denial-of-service state.

Recommendation

Consider using a custom variable (e.g. self-incrementing id_tracker) when assigning ids to campaigns.

3. The token may be locked in the vault after the vault has been deleted

Severity: Low	Category: Business Logic
Target: - contracts/sharedContracts/VotingVault.sol	

Description

contracts/sharedContracts/VotingVault.sol:L36-L42

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

When the Vault's balance is zero, all storage of the contract will be deleted, putting it into a "disabled" state. However, even in this state, it is still possible for tokens to be transferred into the contract (e.g., due to a team error). In such cases, these tokens will be permanently locked in the VotingVault contract.

Recommendation

Consider enabling another feature to allow the manager to withdraw these mis-transferred tokens when disable withdrawTokens() function.

2.3 Informational Findings

4. Missing events for functions that change critical state

Severity: Informational	Category: Logging
Target: - contracts/Periphery/ClaimCampaigns.sol	

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes that allow users to evaluate them. Missing events do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in protocol users.

In the ClaimCampaigns.sol, events are lacking in the privileged setter functions (e.g. changeDonationcollector() function).
contracts/Periphery/ClaimCampaigns.sol:L114-L117

```
function changeDonationcollector(address newCollector) external {
    require(msg.sender == donationCollector);
    donationCollector = newCollector;
}
```

Recommendation

It is recommended to emit events for critical state changes.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 88cf517:

File	SHA-1 hash
ERC721Delegate.sol	d49a3ffd234536c42880b5b1b6960c47c9b257b7
TokenLockupPlans_Bound.sol	c3466673d498ea46107eec090fb12bce225a7c44
VotingTokenLockupPlans_Bound.sol	6b3e6880f2bbae1f68bef2cca8fece0edc5af967
TokenLockupPlans.sol	ba7997659eff13f0ed172b4808ace39614106188
VotingTokenLockupPlans.sol	f5740c5890afff0b2476c465c555d095aea39ff8
BatchPlanner.sol	4d1e7a821e22c1cab1759fed980ba7a0296e161e
ClaimCampaigns.sol	023a2db0e4b497047e30f47ddc22459a49a9963c
TokenVestingPlans.sol	eaa80d06b4223e3b329d788761eb0481c2536195
VotingTokenVestingPlans.sol	f45b4958c3da6e872e0bc9a6437a3c6ddb832017
TimelockLibrary.sol	ff2143be67e8bf82e015e2dd774524f066621b8d
TransferHelper.sol	53ec19ab7cdf3c85f3bf4f4cf12edce67dcda54f
LockupStorage.sol	80e97239a6dc23a6c8c76d637d1b1edfc768efb4
PlanDelegator.sol	c7cbbbf5218e52bbb4144deb5df98d50a5c55086
URIAdmin.sol	f807a67bf13a83a3b3b1e88be61b52710b7707ae
VestingStorage.sol	39bcc4af0471a7ee5e060877d58580fdfe80e606
VotingVault.sol	e063121403f77323c2de332c425c1bcb1725a45f