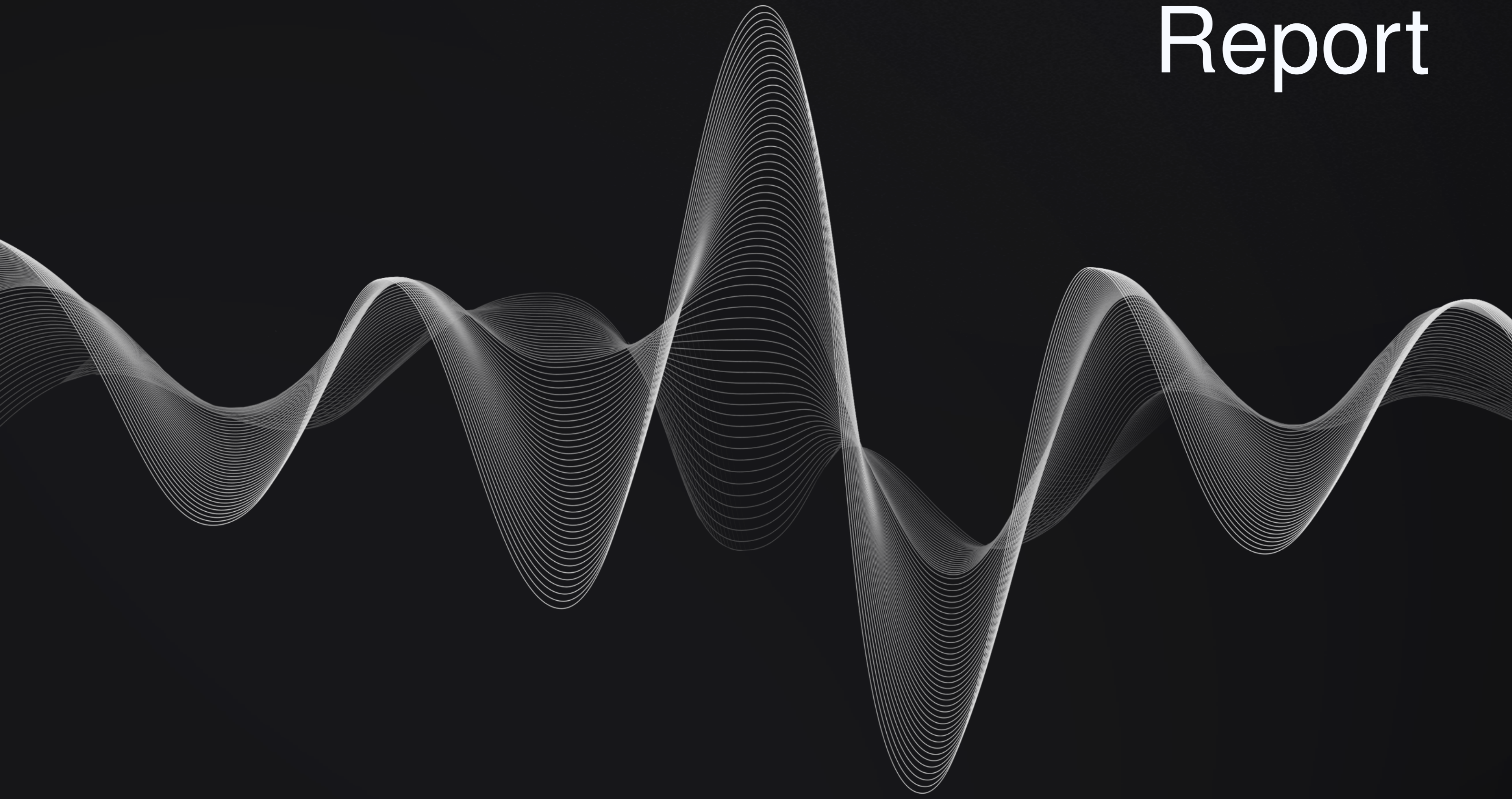
RESONANCE

Hedgey.

Hedgey Finance

Delegated Claims Smart Contract Audit Report



Blockchain, Emerging Technology, and Web2
CYBERSECURITY PRODUCT & SERVICE ADVISORY

Document Control

CONFIDENTIAL **FINAL**(v2.0)

Audit_Report_HDGY-DLG_FINAL_20

Nov 6, 2024	○	v0.1	Luis Arroyo: Initial draft
Nov 6, 2024	○	v0.2	Luis Arroyo: Added findings
Nov 7, 2024	🟢	v1.0	Charles Dray: Approved
Nov 8, 2024	○	v1.1	Luis Arroyo: Reviewed findings
Nov 9, 2024	🟡	v2.0	Charles Dray: Finalized

Points of Contact	Alex Michelsen	Hedgey Finance	alex@hedgey.finance
	Charles Dray	Resonance	charles@resonance.security
Testing Team	Luis Arroyo	Resonance	luis.arroyo@resonance.security
	João Simões	Resonance	joao@resonance.security

Copyright and Disclaimer

The information in this report is considered confidential and proprietary by Resonance and is licensed to the recipient solely under the terms of the project statement of work. Reproduction or distribution, in whole or in part, is strictly prohibited without the express written permission of Resonance.

All activities performed by Resonance in connection with this project were carried out in accordance with the project statement of work and agreed-upon project plan. It's important to note that security assessments are time-limited and may depend on information provided by the client, its affiliates, or partners. As such, the findings documented in this report should not be considered a comprehensive list of all security issues, flaws, or defects in the target system or codebase.

Furthermore, it is hereby assumed that all of the risks in electing not to remedy the security issues identified henceforth are sole responsibility of the respective client. The acknowledgement and understanding of the risks which may arise due to failure to remedy the described security issues, waives and releases any claims against Resonance, now known or hereafter known, on account of damage or financial loss.

Contents

Executive Summary

Hedgey Finance contracted the services of Resonance to conduct a comprehensive security audit of their smart contracts between November 4, 2024 and November 7, 2024. The primary objective of the assessment was to identify any potential security vulnerabilities and ensure the correct functioning of smart contract operations.

During the engagement, Resonance allocated 2 engineers to perform the security review. The engineers, including an accomplished professional with extensive proficiency in blockchain and smart-contract security, encompassing specialized skills in advanced penetration testing, and in-depth knowledge of multiple blockchain protocols, devoted 3 days to the project. The project's test targets, overview, and coverage details are available throughout the next sections of the report.

The ultimate goal of the audit was to provide Hedgey Finance with a detailed summary of the findings, including any identified vulnerabilities, and recommendations to mitigate any discovered risks. The results of the audit are presented in detail further below.

System Overview

Hedgey Finance is an on-chain token infrastructure solution. It provides token vesting, lockups, grants and distributions services for investors, community and teams working on projects. It aims at democratizing access to these tools for companies regardless of their size or stage of development. The core set of tools created by Hedgey combine on-chain token streams and periodic release schedules along with administrative controls like revocability or optional governance rights.

Hedgey Finance's solutions are on-chain, meaning that they are governed by smart contracts implementing, among others, Token Vesting, Investor Lockups, Grant Payouts and Token Distribution. Most EVM-based networks are supported, with prime examples being Ethereum, Arbitrum, Optimism, Polygon and Avalanche.

Repository Coverage and Quality

Code	Tests	Documentation
8 / 10	8 / 10	9 / 10

Resonance's testing team has assessed the Code, Tests, and Documentation coverage and quality of the system and achieved the following results:

- The code follows development best practices and makes use of known patterns, standard libraries, and language guides. It is easily readable and uses the latest stable version of relevant components. Overall, **code quality is good**.

- Unit and integration tests are included. The tests cover both technical and functional requirements. Code coverage is 97%. Overall, **tests coverage and quality is good**.

- The documentation includes the specification of the system, technical details for the code, relevant explanations of workflows and interactions. Overall, **documentation coverage and quality is excellent**.

Target

The objective of this project is to conduct a comprehensive review and security analysis of the smart contracts that are contained within the specified repository.

The following items are included as targets of the security assessment:

- Repository: `hedgey-finance/ClaimCampaigns/contracts`
- Hash: 855476f3c5623e8d8a8eb3215c4103a2ee20fa18
- Latest reviewed hash: **1518e2a82a1e008165e20e1e38f7707cc7b8da35**

The following items are excluded:

- External and standard libraries
- Files pertaining to the deployment process
- Financial related attacks

Methodology

In the context of security audits, Resonance's primary objective is to portray the workflow of a real-world cyber attack against an entity or organization, and document in a report the findings, vulnerabilities, and techniques used by malicious actors. While several approaches can be taken into consideration during the assessment, Resonance's core value comes from the ability to correlate automated and manual analysis of system components and reach a comprehensive understanding and awareness with the customer on security-related issues.

Resonance implements several and extensive verifications based off industry's standards, such as, identification and exploitation of security vulnerabilities both public and proprietary, static and dynamic testing of relevant workflows, adherence and knowledge of security best practices, assurance of system specifications and requirements, and more. Resonance's approach is therefore consistent, credible and essential, for customers to maintain a low degree of risk exposure.

Ultimately, product owners are able to analyze the audit from the perspective of a malicious actor and distinguish where, how, and why security gaps exist in their assets, and mitigate them in a timely fashion.

Source Code Review - Solidity EVM

During source code reviews for Web3 assets, Resonance includes a specific methodology that better attempts to effectively test the system in check:

1. Review specifications, documentation, and functionalities
2. Assert functionalities work as intended and specified
3. Deploy system in test environment and execute deployment processes and tests
4. Perform automated code review with public and proprietary tools
5. Perform manual code review with several experienced engineers
6. Attempt to discover and exploit security-related findings
7. Examine code quality and adherence to development and security best practices
8. Specify concise recommendations and action items
9. Revise mitigating efforts and validate the security of the system

Additionally and specifically for Solidity EVM audits, the following attack scenarios and tests are recreated by Resonance to guarantee the most thorough coverage of the codebase:

- Reentrancy attacks
- Frontrunning attacks
- Unsafe external calls
- Unsafe third party integrations
- Denial of service
- Access control issues

- Inaccurate business logic implementations
- Incorrect gas usage
- Arithmetic issues
- Unsafe callbacks
- Timestamp dependence
- Mishandled panics, errors and exceptions

Severity Rating

Security findings identified by Resonance are rated based on a Severity Rating which is, in turn, calculated off the **impact** and **likelihood** of a related security incident taking place. This rating provides a way to capture the principal characteristics of a finding in these two categories and produce a score reflecting its severity. The score can then be translated into a qualitative representation to help customers properly assess and prioritize their vulnerability management processes.

The **impact** of a finding can be categorized in the following levels:

1. Weak - Inconsequential or minimal damage or loss
2. Medium - Temporary or partial damage or loss
3. Strong - Significant or unrecoverable damage or loss

The **likelihood** of a finding can be categorized in the following levels:

1. Unlikely - Requires substantial knowledge or effort or uncontrollable conditions
2. Likely - Requires technical knowledge or no special conditions
3. Very Likely - Requires trivial knowledge or effort or no conditions

			Likelihood	
		Very Likely	Likely	Unlikely
Impact	Strong	Critical	High	Medium
	Medium	High	Medium	Low
	Weak	Medium	Low	Info

Repository Coverage and Quality Rating

The assessment of Code, Tests, and Documentation coverage and quality is one of many goals of Resonance to maintain a high-level of accountability and excellence in building the Web3 industry. In Resonance it is believed to be paramount that builders start off with a good supporting base, not only development-wise, but also with the different security aspects in mind. A product, well thought out and built right from the start, is inherently a more secure product, and has the potential to be a game-changer for Web3's new generation of blockchains, smart contracts, and dApps.

Accordingly, Resonance implements the evaluation of the code, the tests, and the documentation on a score **from 1 to 10** (1 being the lowest and 10 being the highest) to assess their quality and coverage. In more detail:

- Code should follow development best practices, including usage of known patterns, standard libraries, and language guides. It should be easily readable throughout its structure, completed with relevant comments, and make use of the latest stable version components, which most of the times are naturally more secure.

- Tests should always be included to assess both technical and functional requirements of the system. Unit testing alone does not provide sufficient knowledge about the correct functioning of the code. Integration tests are often where most security issues are found, and should always be included. Furthermore, the tests should cover the entirety of the codebase, making sure no line of code is left unchecked.

- Documentation should provide sufficient knowledge for the users of the system. It is useful for developers and power-users to understand the technical and specification details behind each section of the code, as well as, regular users who need to discern the different functional workflows to interact with the system.

Findings

During the security audit, several findings were identified to possess a certain degree of security-related weaknesses. These findings, represented by unique IDs, are detailed in this section with relevant information including Severity, Category, Status, Code Section, Description, and Recommendation. Further extensive information may be included in corresponding appendices should it be required.

An overview of all the identified findings is outlined in the table below, where they are sorted by Severity and include a **Remediation Priority** metric asserted by Resonance's Testing Team. This metric characterizes findings as follows:

- ▂▂▂▂▂ **"Quick Win"** Requires little work for a high impact on risk reduction.
- ▂▃▆▃▂ **"Standard Fix"** Requires an average amount of work to fully reduce the risk.
- ▃▅▇▅▃ **"Heavy Project"** Requires extensive work for a low impact on risk reduction.

RES-01	🟧	Protocol Fees Can Be Bypassed	▂▃▆▃▂	Acknowledged
RES-02	🟧	Insufficient Signature Validation Leads To Possibility Of Claiming From Arbitrary Campaigns	▂▃▆▃▂	Acknowledged
RES-03	🟩	Missing Zero Address Validation Of treasury	▂▃▆▃▂	Resolved
RES-04	🟩	Critical Changes Should Use Two-step Procedure	▂▂▂▂▂	Acknowledged
RES-05	🟩	Excess Fees Sent Via msg.value Not Refunded	▂▃▆▃▂	Acknowledged
RES-06	🟩	safeTransferFrom() Should Be Used In Place Of transferFrom()	▂▂▂▂▂	Acknowledged
RES-07	⬜	++i/i++ Should Be unchecked{++i}/unchecked{i++}	▂▂▂▂▂	Acknowledged
RES-08	⬜	Non-external Function Names And Variables (Private Or Internal) Should Begin With An Underscore	▂▂▂▂▂	Resolved
RES-09	⬜	Variables Are Initialized With Default Values	▂▂▂▂▂	Resolved

Protocol Fees Can Be Bypassed

Medium **RES-HDGY-DLG01** Data Validation **Acknowledged**

Code Section

- `contracts/ClaimCampaigns.sol#L160`

Description

The protocol adds a fee in native ETH to users through the frontend. However, it is possible to prevent the fee payment by using the smart contract directly without using the web application.

Recommendation

If you want to obtain a commission for the use of the protocol, it is advisable to correctly implement the mechanisms that allow to collect this commission, for example, set the amount to receive and compare it with the msg.value.

Status

> *The issue was acknowledged by Hedgey's team. The development team stated "We acknowledge the issue of having the fee control in the frontend".*

Insufficient Signature Validation Leads To Possibility Of Claiming From Arbitrary Campaigns

Medium **RES-HDGY-DLG02** Business Logic **Acknowledged**

Code Section

- `contracts/ClaimCampaigns.sol#L335-L375`

Description

The function `claimMultipleWithSig()` is used by paymasters to claim tokens from multiple campaigns on behalf of a claimer. It uses a signature verification mechanism to control the access to this function and the ability to claim the claimer's tokens.

The signature provided as an input parameter to this function includes several variables necessary to prevent signature attacks, e.g. signature replay, however, it does not do so effectively. When attempting to claim from multiple campaigns, the paymaster needs to provide a signature that, along with other variables, will verify the id and amount of tokens of the first campaign as well as the number of campaigns to claim. Solely relying on these variables opens up the possibility for the paymaster to claim unintended campaigns.

As an example of a proof of concept:

1. Claimer can claim from 4 campaigns;
2. Claimer provides a proper signature to the paymaster to claim on his behalf for only 3 of the campaigns, specifically campaigns 1, 2, and 3.
3. Paymaster uses the same signature to claim from campaigns 1, 2, and 4.

The scenario identified previously is possible due to the fact that the first campaign matches what the claimer intended, and the number of claimed campaigns also matches. However, it does not match the intended campaigns that were meant to be claimed on behalf of the claimer.

Recommendation

It is recommended to revise the signature verification mechanisms to either include a hash of all the campaign ids, amounts and lengths, or include individual signatures for each of the campaigns to be claimed.

Status

> *The issue was acknowledged by Hedgey's team. The development team stated "We acknowledge the issue but given the complexity and gas costs to resolve, and the fact that we do not see this issue ever arising in real life, we have chosen to leave the signature as is.".*

Missing Zero Address Validation Of treasury

Low **RES-HDGY-DLG03** Data Validation **Resolved**

Code Section

- `contracts/ClaimCampaigns.sol#L165`

Description

The `ClaimCampaigns()` constructor and `changeTreasury()` functions do not validate `_treasury` or `_newTreasury` parameters, so callers can accidentally set important state variables to the zero address.

This could lead to loss of fees.

Recommendation

It is recommended to perform a validation against the Zero Address to ensure future usage of the relevant variables are both handled properly during external calls, and do not result in unintended behavior within the protocol.

Status

The issue has been fixed in commit 1518e2a82a1e008165e20e1e38f7707cc7b8da35.

Critical Changes Should Use Two-step Procedure

Low **RES-HDGY-DLG04** Data Validation **Acknowledged**

Code Section

- `contracts/ClaimCampaigns.sol#L165`

Description

The critical procedures used to change treasury or whitelist or blacklist tokens in this protocol should use a two step process in order to ensure the modification is indeed intended.

Recommendation

The lack of two-step procedures for critical operations leaves them prone to mistakes. Consider adding two-step procedures on critical functions, such as, `changeTreasury()`.

Status

> *The issue was acknowledged by Hedgey's team. The development team stated "we'll just leave it as is for sake of ease, rather than add more functions to account for ability to try and confirm the updated address - since it is all internally managed.".*

Excess Fees Sent Via msg.value Not Refunded

Low **RES-HDGY-DLG05** Data Validation **Acknowledged**

Code Section

- `contracts/ClaimCampaigns.sol#L160`

Description

The smart contract allows users to provide ETH as fee, but does not refund the amount in excess of what is required, sending incorrect amounts to treasury.

Recommendation

The protocol should refund the excess fees that users may place by mistake.

Status

> *The issue was acknowledged by Hedgey's team. The development team stated "the front end app controls the entire msg.value and no way for the smart contract to know how much fees should be sent in the msg.value without adding an additional 'fee' param which can be messed with and create more issues".*

safeTransferFrom() Should Be Used In Place Of transferFrom()

Low | **RES-HDGY-DLG06** | Data Validation | **Acknowledged**

Code Section

- `contracts/ClaimCampaigns.sol#L694`
- `contracts/ClaimCampaigns.sol#L708`

Description

Unlike the basic Transfer function, SafeTransfer incorporates safeguards against potential smart contract vulnerabilities, such as reentrancy attacks and unexpected token loss.

Recommendation

SafeTransfer should be used in place of Transfer for Solidity contracts to ensure robust security and error handling.

Status

> *The issue was acknowledged by Hedgey's team. The development team stated "the actual safeTransferFrom() function is overriden to revert, so I have to keep that as a transferFrom()".*
>
> *However, the team updated the function for the transferable tokenlockups.*

++i/i++ Should Be unchecked{++i}/unchecked{i++}

Info **RES-HDGY-DLG07** Gas Optimization **Acknowledged**

Code Section

- `contracts/ClaimCampaigns.sol#L143`
- `contracts/ClaimCampaigns.sol#L245`

Description

When it is not possible for counters to overflow, as is the case when used in for and while-loops, they should be `unchecked`. The `unchecked` keyword is new in solidity version 0.8.0, so this only applies to that version or higher, which these instances are. This saves 30-40 gas per loop.

Recommendation

Consider increment the counters in `unchecked` code in order to save gas.

Status

The issue was acknowledged by Hedgey's team. The development team stated "Acknowledged, but will leave for consistency of codebase".

Non-external Function Names And Variables (Private Or Internal) Should Begin With An Underscore

Info **RES-HDGY-DLG08** Code Quality **Resolved**

Code Section

- `contracts/ClaimCampaigns.sol`

Description

According to the Solidity Style Guide, non-external functions and variable names should begin with an underscore.

Recommendation

Consider renaming the affected private variables and functions.

Status

The issue has been fixed in commit 1518e2a82a1e008165e20e1e38f7707cc7b8da35.

Variables Are Initialized With Default Values

Info **RES-HDGY-DLG09** Gas Optimization **Resolved**

Code Section

- `contracts/ClaimCampaigns.sol#L143`
- `contracts/ClaimCampaigns.sol#L245`

Description

Index in for-loops are initialized to 0, which is not necessary.

Recommendation

It is recommended to avoid initializing variables to its default value.

Status

The issue has been fixed in commit 1518e2a82a1e008165e20e1e38f7707cc7b8da35.

Proof of Concepts

No Proof-of-Concept was deemed relevant to describe findings in this engagement.