SALUS SECURITY

APR 2025

CODE SECURITY ASSESSMENT

PAY2REACH

Overview

Project Summary

- Name: Pay2reach
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
  - https://github.com/Pay2ReachDev/PayToReachContract
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Pay2reach
Version	v2
Type	Solidity
Dates	Apr 16 2025
Logs	Apr 14 2025; Apr 16 2025

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	3
Total Low-Severity issues	2
Total informational issues	2
Total	7

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	Custom tokens can be used for payments	Medium	Business Logic	Resolved
2	Even if answered after the deadline a fee will still be charged	Medium	Business Logic	Resolved
3	Centralization risk	Medium	Centralization	Acknowledge
4	Multiple spendings may occur under certain circumstances	Low	Business Logic	Resolved
5	Missing events for functions that change critical state	Low	Logging	Resolved
6	Missing two-step transfer ownership pattern	Informational	Business Logic	Acknowledge
7	Use of floating pragma	Informational	Configuration	Resolved

2.2 Notable Findings

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

1. Custom tokens can be used for payments	
Severity: Medium	Category: Business Logic
Target: - contracts/diamond/facets/Pay2ReachOrderFacet.sol	

Description

When creating an order, the sender is required to prepay tokens, which will be transferred upon receiving an answer. Since the `` `_token` `` parameter is user-defined during order creation, an attacker could create a malicious token that only ensures `` `transfer` `` and `` `transferFrom` `` do not revert. As a result, after an answer is provided, the actual token transferred could be a worthless token with no value.

contracts/diamond/facets/Pay2ReachOrderFacet.sol:L14 - L43

```
function createOrder(
    uint256 _id,
    string memory _senderSocialMediaId,
    string memory _kolSocialMediaId,
    uint256 _amount,
    address _token,
    uint256 _deadline
) external nonReentrant {
    ...

    s.orders[_id] = LibAppStorage.Order({
        ...
        token: _token,
        ...
    });

    _collectOrderTokens(_id, _token, _amount);
}
```

Recommendation

Consider implementing measures such as adding a whitelist for acceptable token addresses to prevent the use of malicious or worthless tokens.

Status

The team has resolved this issue in commit 250b812.

2. Even if answered after the deadline a fee will still be charged	
Severity: Medium	Category: Business Logic
Target: - contracts/diamond/facets/Pay2ReachOrderFacet.sol	

Description

The `answerOrder` function does not check whether the order has expired. As a result, even if the order is answered after the deadline, a fee will still be charged. This behavior may be inappropriate for time-sensitive messages — in such cases, the system should not charge a fee but simply update the order's status to `Expired`.

Recommendation

Consider not transferring tokens to KOL when an expired order is answered.

Status

The team has resolved this issue in commit 250b812.

3. Centralization risk	
Severity: Medium	Category: Centralization
Target: - contracts/diamond/facets/Pay2ReachPayFacet.sol - contracts/diamond/facets/Pay2ReachOrderFacet.sol - contracts/diamond/facets/OwnershipFacet.sol - contracts/diamond/facets/DiamondCutFacet.sol	

Description

The contract includes a privileged address, `owner`, which has the authority to modify critical parameters such as `responseTimeLimit` and `platformFee`. The owner can also call functions like `refund` and `payTokens` to extract funds from orders, as well as `collectTokens` to transfer tokens previously approved by users.

If the owner's private key is compromised, an attacker could exploit these privileged operations to harm both the project and its users.

If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the other users.

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team.

4. Multiple spendings may occur under certain circumstances	
Severity: Low	Category: Business Logic
Target: - contracts/diamond/facets/Pay2ReachPayFacet.sol	

Description

In both the `cancelOrder()` and `refundTokens()` functions, the `order.amount` is not updated. When the owner calls `cancelOrder()` or `refundTokens()`, the funds will be transferred, but the amount is not deducted. Under certain conditions, the owner may still be able to call `refundTokens()` to perform a refund.

contracts/diamond/facets/Pay2ReachPayFacet.sol:L94 - L135

```
function refundTokens(
        uint256 _orderId,
        address _sender,
        uint256 _fee
    ) external onlyOwnerOrSelf {
        ...
        require(
            order.status == LibAppStorage.OrderStatus.Pending ||
                order.status == LibAppStorage.OrderStatus.Cancelled,
            "Order must be pending or cancelled for refund"
        );

        require(_fee < order.amount, "Fee is greater than amount");

        uint256 amount = order.amount - _fee;

        ...
    }
```

Recommendation

It is recommended to change the state of the `order.amount` in the `refundTokens()` function.

Status

The team has resolved this issue in commit 250b812.

5. Missing events for functions that change critical state	
Severity: Low	Category: Logging
Target: - contracts/diamond/facets/Pay2ReachPayFacet.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 `Pay2ReachPayFacet` contract, events are lacking in the `setFeeRecipient()` function.

Recommendation

It is recommended to emit events for critical state changes.

Status

The team has resolved this issue in commit 250b812.

2.3 Informational Findings

6. Missing two-step transfer ownership pattern	
Severity: Informational	Category: Business logic
Target: - contracts/diamond/LibDiamond.sol	

Description

The `LibDiamond` library uses a custom function `setContractOwner()` which is a simple mechanism to transfer the ownership not supporting a two-step transfer ownership pattern. This simpler mechanism can be useful for quick tests, but projects with production concerns are likely to outgrow it. Transferring ownership is a critical operation and this could lead to transferring it to an inaccessible wallet or renouncing the ownership, e.g. mistakenly.

Recommendation

It is recommended to implement a two-step transfer of ownership mechanism where the ownership is transferred and later claimed by a new owner to confirm the whole process and prevent lockout.

Status

This issue has been acknowledged by the team.

7. Use of floating pragma	
Severity: Informational	Category: Configuration
Target: - All	

Description

```
pragma solidity ^0.8.28;
```

All contracts use a floating compiler version `^0.8.28`.

Using a floating pragma `^0.8.28`statement is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

Status

The team has resolved this issue in commit 250b812.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 28b60cd:

File	SHA-1 hash
contracts/Pay2ReachDiamond.sol	5447c157fc087cc27ac21cde1d1c655acbb32cd8
contracts/diamond/facets/DiamondCutFacet.sol	f1c5e0f1e21df0e06a732f47ed48871e070432e8
contracts/diamond/facets/DiamondLoupeFacet.sol	aaf4c5dec753e5526bd6a7e2aebdc8a8e089323e
contracts/diamond/facets/OwnershipFacet.sol	6d649ede1bfa78d9aba660d5c7560d20e735ba0e
contracts/diamond/facets/Pay2ReachOrderFacet.sol	0c78284a78f7d7d95ffd50f870fc04ad8968b81f
contracts/diamond/facets/Pay2ReachPayFacet.sol	36bf5c30c9266b40ad58ae10c40cc6ad73eceb60
contracts/diamond/libraries/LibAppStorage.sol	3e88fe681718b62f5faa8c313267d6188c09f2c0