Liquify

Referral Manager Upgrade Audit Report

Version 1.0

Conducted by:
Alin Barbatei (ABA) — Lead Silverologist

April 25, 2025

Table of Contents

1 Introduction

1.1 About The Auditors

ABA, or Alin Mihai Barbatei, is an established independent security researcher with deep expertise in blockchain security. With a background in traditional information security and competitive hacking, ABA has a proven track record of discovering hidden vulnerabilities. He has extensive experience in securing both EVM (Ethereum Virtual Machine) compatible blockchain projects and Bitcoin L2, Stacks projects.

Having conducted several solo and collaborative smart contract security reviews, ABA consistently strives to provide top-quality security auditing services. His dedication to the field is evident in the top-notch, high-quality, comprehensive smart contract auditing services he offers.

To learn more about his services, visit ABA's website abarbatei.xyz. You can also view his audit portfolio here. For audit bookings and security review inquiries, you can reach out to ABA on Telegram, Twitter (X) or WarpCast:

`https://t.me/abarbatei`

`https://x.com/abarbatei`

`https://warpcast.com/abarbatei.eth`

For this audit, Silverologist (@silverologist), a promising auditor, contributed their expertise to help secure the protocol.

1.2 About Liquify Referral Manager

In the broader ecosystem Liquify Ventures ecosystem, the Referral Manager is a key component used to store and distribute referral bonuses for all intended market participants.

The audited smart contract is intended to be applied as an upgrade to the already-deployed `ReferralManager` contract with the added feature of being able to increase the referral bonus for specific addresses.

This feature acts as a mechanism for reducing intended targets general fee and may help with capital acquisition and retention of larger investors.

1.3 Issues Risk Classification

The current report contains issues, or findings, that impact the protocol. Depending on the likelihood of the issue appearing and its impact (damage), an issue is in one of four risk categories or severities: *Critical*, *High*, *Medium*, *Low* or *Informational*.

The following table shows an overview of how likelihood and impact determines the severity of an issue.

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behavior that's not so critical.

Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

Informational findings

Informational findings encompass recommendations to enhance code style, operations alignment with industry best practices, gas optimizations, adherence to documentation, standards, and overall contract design.

Informational findings typically have minimal impact on code functionality or security risk.

Evaluating all vulnerability types, including informational ones, is crucial for a comprehensive security audit to ensure robustness and reliability.

2 Executive Summary

2.1 Overview

Project Name	Liquify Referral Manager Upgrade
Codebase	https://github.com/Liquify-labs/sc
Operating platform	Arbitrum, Unichain, Polygon, Base, Optimism, Ethereum, Berachain, Sonic, Soneium, BSC
Programming language	Solidity
Initial commit	`f52bb19f85454a44a490217ff619c8e661d9a33b`
Remediation commit	`6d1e29b52b64beef253a489bd65573cdb0548ece`
Timeline	From 05.06.2025 to 08.06.2025 (4 days)
Audit methodology	Static analysis and manual review

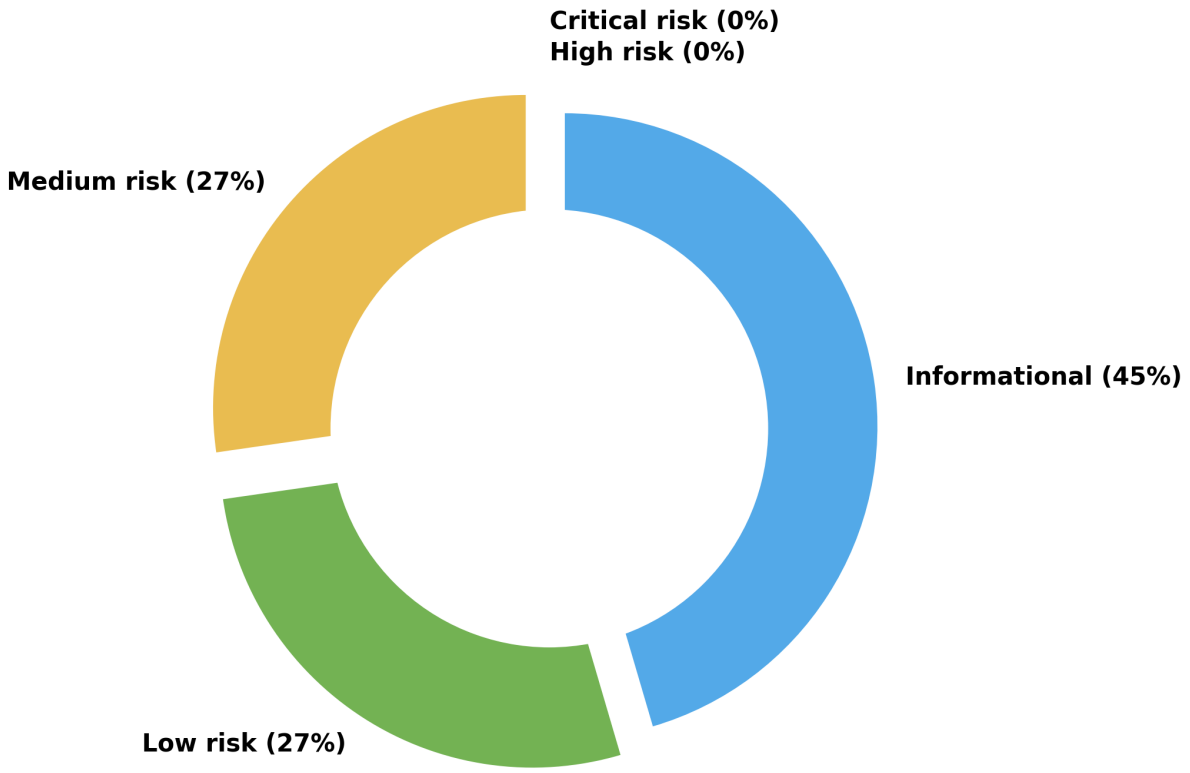
2.2 Audit Scope

Files and folders in scope

- `src/ReferralManagerV2.sol`
- `src/interfaces/IReferralManager.sol`

2.3 Summary of Findings

Severity	Total Found	Resolved	Partially Resolved	Acknowledged
Critical risk	0	0	0	0
High risk	0	0	0	0
Medium risk	3	1	0	2
Low risk	3	0	0	3
Informational	5	0	0	5



2.4 Findings & Resolutions

ID	Title	Severity	Status
M-01	Referral Manager fee value handling design may cause core contribution issues	Medium	Acknowledged
M-02	Incorrect cumulative referral fee calculation	Medium	Resolved
M-03	Self-referral can be abused to always ensure a discount	Medium	Acknowledged
L-01	Cannot initialize contract if the gnosis wallet parameter is not the caller	Low	Acknowledged
L-02	ReferrerBalancesUpdated event emission has incorrect arguments order	Low	Acknowledged
L-03	Setting a special fee overrides existing referrers	Low	Acknowledged
I-01	_processSpecialReferral function can be simplified	Informational	Acknowledged
I-02	Adding or removing Liquify contracts lacks validations	Informational	Acknowledged

ID	Title	Severity	Status
I-03	Add security contact to contract	Informational	Acknowledged
I-04	Miscellaneous contract improvements	Informational	Acknowledged
I-05	Setting referrers related data does not validated initiator	Informational	Acknowledged

3 Findings

3.1 Medium Severity Findings

[M-01] Referral Manager fee value handling design may cause core contribution issues

> **Severity:** *Medium risk* *(Acknowledged)*
>
> **Context:** *Liquify.sol ReferralManagerV2.sol*

Description

The `ReferralManagerV2` contract is designed to store and calculate per user fees percentages out of the input amount, when asked to process the fees via the `processReferralFees` function to also require an amount for which to apply the fee percentage.

This design may cause issues with integrating `Liquify` core contracts when setting the protocol fee percentage. The `Liquify` contract checks that it does not fall below the total referral fees defined in the referral manager.

However, referral fees can be increased independently afterwards, without any verification that they remain within the bounds of any existing core contract protocol fees.

Consider the following scenario:

- `Liquify::protocolFeeBPS` is set to 5%
- `ReferralManagerV2::getTotalReferralsFeesInBPS` initially returns 5%
- The referral manager's owner updates the referral fees to a total of 6%

This situation is currently allowed and leads to the protocol fee being insufficient to cover the full referral fee distribution. Implicitly, this results in core contributions being blocked as `Liquify::contribute` calls revert with an underflow.

Recommendation

With the current design of the Referral Manger, the issue cannot be resolved, only mitigated by ensuring off-chain that all Liquify core contracts that use the referral manager keep the protocol fee above the referral manager cumulative fee, and any fee changes within the referral manager needs to first ensure it will not cause any issues with existing core contracts.

To resolve the issue, referral manager needs to be changed as follows:

- noted fee percentage should become as percentages of the actual protocol fee amount, not out of the input amount.
  - example, for a 5% referral fee (out of the input amount), when the core fee is capped at 10%, means that the referral fee is 50% (50% of 10% = 5%)

- have the `processReferralFees` function be called with the fee amount, not the input amount, from the Liquify core contracts, and work with the new logic

Resolution: Acknowledged by the team.

[M-02] Incorrect cumulative referral fee calculation

> **Severity:** *Medium risk* *(Resolved)*
>
> **Context:** ***ReferralManagerV2.sol***:*120,248*

Description

The `ReferralManagerV2` fee processing logic allows users to supply their own referrer alongside the protocol noted referrer.

The fee percentage amount between all referrer types altogether are validated when set via the `setReferrerFees` function.

During validation, the sum of all four referral-related fees (`_referrerInitiatorFeeBPS`, `_superReferrerInitiatorFeeBPS`, `_referrerContributionFeeBPS`, and `_superReferrerContributionFeeBPS`) is checked against `MAX_REFERRER_BPS`, implying that each fee is applied independently and cumulatively.

```
uint16 totalBps = _referrerInitiatorFeeBPS + _superReferrerInitiatorFeeBPS +
    _referrerContributionFeeBPS + _superReferrerContributionFeeBPS;
if (totalBps > MAX_REFERRER_BPS) {
    revert CombinedReferralFeesExceedLimit();
}
```

However, in the `_processReferral` function, the super referrer fee is deducted from the corresponding referrer fee, meaning the total applied fee is actually lower than the sum checked during validation.

```
uint256 referrerFee = amount * referrerFeeBPS / MAX_BPS - superReferrerFee;
```

For example, if all four fees are set to:

- `referrerInitiatorFeeBPS` = 1%
- `superReferrerInitiatorFeeBPS` = 0.1%
- `referrerContributionFeeBPS` = 1%
- `superReferrerContributionFeeBPS` = 0.1%

The validation logic checks that the total of `2.2%` is under the cap, but the actual fees applied are `(0.9% + 0.1%) + (0.9% + 0.1%) = 2%`.

Example, the following combination has a practical total fee of 5% (`referrerInitiatorFeeBPS (2.5%)` + `referrerContributionFeeBPS (2.5%)`)

- `referrerInitiatorFeeBPS` = 2.5%
- `superReferrerInitiatorFeeBPS` = 1%

- `referrerContributionFeeBPS` = 2.5%
- `superReferrerContributionFeeBPS` = 1%

but when set via `setReferrerFees`, it is counted as 7% and not allowed.

This mismatch between validation and application leads to the `ReferralManagerV2` contract not accepting some valid fee combinations.

Recommendation

Update the fee validation to reflect how fees are actually applied. Specifically, only include `_referrerInitiatorFeeBPS` and `_referrerContributionFeeBPS` in the total for validation.

The same change must also be applied to the `getTotalReferralsFeesInBPS` function.

Resolution: Resolved, the recommended fix was implemented in **#6d1e29b**.

[M-03] Self-referral can be abused to always ensure a discount

> **Severity:** *Medium risk* *(Acknowledged)*
>
> **Context:** ***Liquify.sol***:504 ***ReferralManagerV2.sol***:183-193

Description

The `ReferralManagerV2` fee processing logic allows users to supply their own referral alongside the protocol noted referrer.

If the user referral (`userReferrer`) is present, the fee calculation will allocate a portion of the protocol fee to that user, as denoted by the `referrerContributionFeeBPS` and `superReferrerContributionFeeBPS` (for the super referrer equivalent) fee variables.

This mechanism is designed to incentivize the protocol community to add more members.

An issue with the current user-supplied referral address is that there is no validation done to it.

Normally, referrals (or affiliate links) are generated by the product for specific individuals which then allow other to use the product, while providing the specific referral token.

However, in the case of Liquify and the Referral Manager, users that contribute to a project via `Liquify::contribute`, can pass any `userReferrer` which simply allocates a part of the protocol's fee to that address.

Any contributor can always self-referral to ensure he always gets a discount. This both reduces protocol revenue and disincentivizes legitimate protocol users to use the mechanism in an ethical manner.

Recommendation

Create a whitelisting in the `ReferralManagerV2` contract where the protocol team can store which users are allowed to be referenced. Then, during a `processReferralFees` call, check and only allow user referrals (or super referrals) that are whitelisted.

Resolution: Acknowledged by the team.

ABA: the team is aware of this and considers it an acceptable tradeoff regarding their referral design in order to maintain it as simple as possible.

3.2 Low Severity Findings

[L-01] Cannot initialize contract if the gnosis wallet parameter is not the caller

> **Severity:** *Low risk (Acknowledged)*
>
> **Context:** *ReferralManagerV2.sol:71-77*

Description

The `ReferralManagerV2::initialize` function takes an `gnosisWallet` address parameter that is set as the contract owner using OpenZeppelin's `Ownable` implementation.

The `initialize` function then calls `setReferrerFees`, a function which has the `onlyOwner` modifier.

This leads to the contract initialization failing if `msg.sender != gnosisWallet`.

Note, as the contract has already been deployed and the `initialize` function called, this issue can no longer manifest on the live version. It can only manifest on subsequent referral manger deployments.

Recommendation

Move the current functionality of `setReferrerFees` into an internal `_setReferrerFees` version, without the `onlyOwner` modifier and call that version from the `initialize` function. The `setReferrerFees` public version is then changed to `external` and just calls the internal version.

If the original intent is to only deploy a referral manager via the team gnosis wallet, then remove the `gnosisWallet` argument and use `msg.sender` for ownable initialization.

Resolution: Acknowledged by the team.

[L-02] ReferrerBalancesUpdated event emission has incorrect arguments order

> **Severity:** *Low risk (Acknowledged)*
>
> **Context:** *ReferralManagerV2.sol:196-204*

Description

After the `ReferralManagerV2::processReferralFees` function executes, the `ReferrerBalancesUpdated` event is emitted.

```
event ReferrerBalancesUpdated(
    address indexed sender,
    address indexed projectOwner,
    address indexed paymentToken,
    uint256 initiatorSuperReferrer,
```

```
    uint256 initiatorReferrer,
    uint256 userSuperReferrer,
    uint256 userReferrer
);
```

However, the emission incorrectly uses the `super` referrer balance instead of the simple referrer balance, in both the user and initiator pairing:

```
emit ReferrerBalancesUpdated(                                    // event documentation
    sender,                                                      // sender
    initiator,                                                   // projectOwner
    address(paymentToken),                                       // paymentToken
    referrerBalances[referrers.referrer][paymentToken],          //
        initiatorSuperReferrer
    referrerBalances[referrers.superReferrer][paymentToken],// initiatorReferrer
    referrerBalances[userReferrer][paymentToken],                // userSuperReferrer
    referrerBalances[superReferrer][paymentToken]                // userReferrer
);
```

This results in incorrect off-chain balance monitoring.

Recommendation

Change `ReferrerBalancesUpdated` emission so that `referrers.superReferrer` is interchanged with `referrers.referrer` and `superReferrer` is interchanged `userReferrer`.

Resolution: Acknowledged by the team.

ABA: team is aware of the issue but has chosen to resolve it off-chain.

[L-03] Setting a special fee overrides existing referrers

> **Severity:** ***Low risk*** *(Acknowledged)*
>
> **Context:** ***ReferralManagerV2.sol****:102-112*

Description

The new `ReferralManagerV2` contract added a special initiator fee that takes priority over other existing fees and can be up to the protocol fee itself.

When this status is given via a `setSpecialFee` call, the provided `initiator` address is both set as initiator and self referrer.

However, there is no validation that for the existing initiator there isn't a referrer already set, similar to how this check exists in the `setReferrers` function.

This allows bypassing the constraint that once a referrer is set, it remains as so.

Recommendation

In the `ReferralManagerV2::setSpecialFee` function, apply the same check as in the setReferrers function to ensure the referrer is not already set:

```
if (initiatorToReferrers[initiator].referrer != address(0)) revert
    ReferrerAlreadySet();
```

Resolution: Acknowledged by the team.

3.3 Informational Findings

[I-01] _processSpecialReferral function can be simplified

> **Severity:** *Informational* *(Acknowledged)*
>
> **Context:** *ReferralManagerV2.sol*:254-278

Description

The internal `_processSpecialReferral` function is used to determine the special referral fee value.

This function has several redundancies and can be written in a more concise and less gas intensive manner.

- the function is called only when there is a custom fee, via a `if(initiatorFeeBPS[initiator] > 0)` check, thus doing the check again in `_processSpecialReferral` is redundant, remove it.
- `_processSpecialReferral` uses an unnamed return variable and does not explicitly return in all code paths. Since it returns `feeAmount` in all cases (or 0) but after `feeAmount` is used, a named `feeAmount` can be used to simplify the code and the double `uint256 feeAmount` declaration can be removed.
- both if-else branches of the `if(overrideBPS > protocolFeeBPS)` condition differentiate by which fee value is used only. The `referrerBalances` update is identical and can be move outside the `if` conditional branch altogether
- since the `feeAmount` in the if-else branch is differentiated only if using the `protocolFeeBPS` or `overrideBPS`, consider using a ternary (`?`) if clause to only differentiate that fee and then using it as intended.

Recommendation

Apply the mentioned changes to simplify and de-clutter the code.

Resolution: Acknowledged by the team.

[I-02] Adding or removing Liquify contracts lacks validations

> **Severity:** *Informational* *(Acknowledged)*
>
> **Context:** *ReferralManagerV2.sol*:83-91

Description

In the `ReferralManagerV2` contract, approved `Liquify` core contracts can be added via the `addLiquifyContract` function and removed via the `removeLiquifyContract` function.

Neither of these functions apply any input validation for the address that is passed:

- in `addLiquifyContract`
  - there is no check that the passed address is not already approved
  - there us no check to ensure address 0 cannot be passed
- in `removeLiquifyContract`
  - there is no check that the passed address is not already removed
  - there us no check to ensure address 0 cannot be passed

Recommendation

Add the missing checks in the mentioned functions.

Resolution: Acknowledged by the team.

[I-03] Add security contact to contract

> **Severity:** *Informational* *(Acknowledged)*
>
> **Context:** *ReferralManagerV2.sol*

Description

In case a whitehat identifies an issue with the on-chain contracts of the protocol, to more easily be able to contact the team, have a security contact added to the contract.

Recommendation

Add the email address for the team security contact either as a plain comment or as a custom:security-contact natspec tag to the contract.

Resolution: Acknowledged by the team.

[I-04] Miscellaneous contract improvements

> **Severity:** *Informational* *(Acknowledged)*
>
> **Context:** *ReferralManagerV2.sol:206,102*

Description

Throughout the contract there are several small improvements that can be made to the codebase.

In the `ReferralManagerV2::processReferralFees` improvements can be made:

1. the `// 2) see if they have a custom override` comment starts with a `2)` while there is no initial `1)` check. Remove the `2)` ordering
2. when calling the `_processReferral` function, the `paymentToken` and `amount` arguments are not on the same indent level as the other arguments. Bring them to the same indent level.

3. returning the `totalFee` amount at the end of the function is redundant as it is a named return variable, meaning that it will be returned regardless.

In the `setSpecialFee` function, the `bps` parameter can be changed to a more meaningful name, such as `feeBps`.

Recommendation

Apply the mentioned changes to improve the code quality of the `processReferralFees` and `setSpecialFee` functions.

Resolution: Acknowledged by the team.

[I-05] Setting referrers related data does not validated initiator

> **Severity:** *Informational* *(Acknowledged)*
>
> **Context:** *ReferralManagerV2.sol:93-112*

Description

When the `ReferralManagerV2` sets referrers via the `setReferrers` function or a special fee via a `setSpecialFee` call, there is no validation that the passed `initiator` is not the zero address.

If, by mistake `address(0)` is passed when calling these function, the zero address would then be set as having referees.

This does not result in any meaningful issue, as there are no means of increasing any initiator balance outside of `processReferralFees` and this function is always called with a non-zero address value from `Liquify` core.

Adding the check here helps to flag mistakes easier in team protocol configurations.

Recommendation

Add a zero address check in the `setReferrers` and `setSpecialFee` functions.

Resolution: Acknowledged by the team.

4 Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts the consultant to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. The consultant’s position is that each company and individual are responsible for their own due diligence and continuous security. The consultant’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology that is analyzed.

The assessment services provided by the consultant is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. Furthermore, because a single assessment can never be considered comprehensive, multiple independent assessments paired with a bug bounty program are always recommend.

For each finding, the consultant provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved, but they may not be tested or functional code. These recommendations are not exhaustive, and the clients are encouraged to consider them as a starting point for further discussion.

The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties. Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, the consultant does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

The consultant retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. The consultant is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. The consultant is furthermore allowed to claim bug bounties from third-parties while doing so.