Liquify

Unified Bridge Audit Report

Version 1.1

Conducted by:
Alin Barbatei (ABA) — Lead Silverologist

April 24, 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 Unified Bridge

Liquify Ventures created the unified bridge component to serve as an interlocking component in their ecosystem.

At a smart contract level, its purpose is to permit initiators to deposit tokens which will then be released to contributors over time and under specific conditions.

In the broader ecosystem, it allows cross-chain token distribution with support for all Liquify type projects.

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 Unified Bridge
Codebase	https://github.com/Liquify-labs/sc-liquify-bridge
Operating platform	Arbitrum, Unichain, Polygon, Base, Optimism, Ethereum, Berachain, Sonic, Soneium, BSC
Programming language	Solidity
Audited commits	[1] a54378a9e6274792d8ec8e7486016c3030578e1c [2] e01a2de205fd5f8ccdac4ca0e2fe5e23fb037130
Remediation commit	7f97f05f9e192c4372ab55348ac326e9523c408a
Audit methodology	Static analysis and manual review

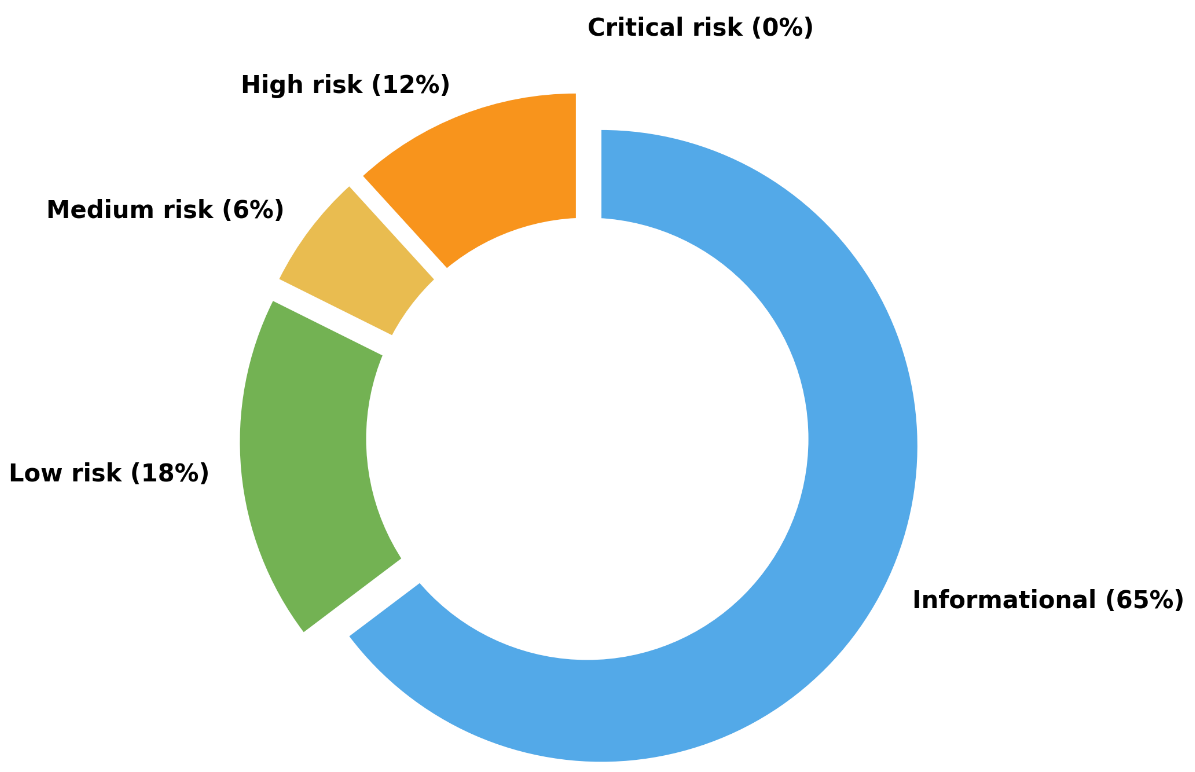
2.2 Audit Scope

Files and folders in scope

```
- contracts/LiquifyBridge.sol
- contracts/LiquifyUnifiedBridge.sol
- contracts/interfaces/ILiquifyBridge.sol
```

2.3 Summary of Findings

Severity	Total Found	Resolved	Partially Resolved	Acknowledged
Critical risk	0	0	0	0
High risk	2	2	0	0
Medium risk	1	0	0	1
Low risk	3	1	0	2
Informational	11	7	2	2



2.4 Findings & Resolutions

ID	Title	Severity	Status
H-01	Insufficient deposit checks lead to over withdrawal	High	Resolved
H-02	Initiator can decide token price	High	Resolved
M-01	Protocol does not support rebase or fee-on-transfer tokens	Medium	Acknowledged
L-01	Cannot deploy when owner is not deployer	Low	Resolved
L-02	Setup signatures don't expire	Low	Acknowledged
L-03	Dust is lost on deposits due to rounding	Low	Acknowledged
I-01	Event parameter inconsistencies	Informational	Resolved
I-02	Storage handling ca be simplified	Informational	Resolved
I-03	Remove debug remnants	Informational	Acknowledged
I-04	Missing events on significant action	Informational	Resolved
I-05	Use a two-step ownership transfer routine	Informational	Resolved

ID	Title	Severity	Status
I-06	Add security contact to contracts	Informational	Resolved
I-07	getVestingReleaseAmounts and getProjectVestingAmounts are identical	Informational	Resolved
I-08	Nonstandard _key view function name	Informational	Resolved
I-09	Bridge contract can be simplified or optimized	Informational	Partially Resolved
I-10	Cleanup codebase	Informational	Partially Resolved
I-11	Missing validation for 18-decimal token assumption	Informational	Acknowledged

3 Findings

3.1 High Severity Findings

[H-01] Insufficient deposit checks lead to over withdrawal

Severity: *High risk* *(Resolved)*

Context: [1]: ***LiquifyBridge.sol***:*80-83,105-107,155-158*

Description

Deposited tokens from an initiator are allocated across four categories:

- Tokens already claimed by users
- Tokens approved for the owner to claim
- Tokens already claimed by the owner
- The remaining unallocated balance available for future user claims or owner approvals

To ensure correct behavior, any operation that consumes from the free balance must validate that the requested amount does not exceed it.

However, this check is not consistently implemented in the contract. The following functions incorrectly omit part of the accounting.

- `approveOwnerToClaim` ignores tokens already claimed by the owner:

```
if (initiatorTokenAllocationState[msg.sender][token].depositedAmount
- initiatorTokenAllocationState[msg.sender][token].claimedAmountByUsers
- initiatorTokenAllocationState[msg.sender][token].approvedAmountToBeClaimedByOwner
    < amount)
    revert InsufficientDepositedAmount();
```

- `claim` ignores tokens approved for or claimed by the owner:

```
if (initiatorTokenAllocationState[initiator][token].depositedAmount -
    initiatorTokenAllocationState[initiator][token].claimedAmountByUsers < amount)
    revert InsufficientDepositedAmount();
```

- `emergencyWithdraw` ignores tokens already claimed by the owner:

```
if (initiatorTokenAllocationState[initiator][token].depositedAmount
    - initiatorTokenAllocationState[initiator][token].claimedAmountByUsers
    < amount)
    revert InsufficientDepositedAmount();
```

This faulty logic allows over-withdrawals. Consider the following scenario:

- `alice` deposits 10 tokens
- bob deposits 10 tokens

- the signer approves `charlie` to claim 10 tokens from bob
- bob approves 5 tokens for the owner to withdraw
- the owner withdraws 5 tokens from bob
- `charlie` claims 10 tokens from bob
- ⇒ Total withdrawn: 15 tokens from bob's balance of 10. The excess 5 tokens effectively come from `alice` or other initiators, violating isolation between depositors.

Recommendation

In the `approveOwnerToClaim` and `emergencyWithdraw` functions, also take into consideration the amount claimed by the owner. For the `claim` function, also consider approved for or claimed by the owner amounts.

Resolution: Resolved. A fix was implemented in **#e01a2de**.

ABA: the deposit-withdraw mechanism has been completely revamped and this issue no longer applies.

[H-02] Initiator can decide token price

Severity: *High risk* *(Resolved)*

Context: [⊸2] : ***LiquifyUnifiedBridge.sol***:*182-206*

Description

When a user configures a project for deposit on the `LiquifyUnifiedBridge`, alongside the input he passes in the call, he also receives a signature, signed by the protocol to validate the input data.

This signature, incorrectly, only validates part of the input data. Specifically, the token `price` is left out.

The token `price` is used to calculate the `totalSupply`, which represents the total amount of tokens the user needs to deposit, cumulative for each vesting period.

```
uint256 totalSupply = (raised * 1e18) / price;
```

Since initiators can decide their own price, they can also arbitrary decide the amount of tokens they are willing to pay to users, which can leave a loss to users.

An ill intended initiator can separate the majority of his tokens in the last vesting period or have only one vesting period and refuse to pay users a fair price.

Recommendation

Include the token `price` in the `Setup` signature.

Resolution: Resolved, the recommended fix was implemented in **#89dd5ff** and **#d5d34b9**.

3.2 Medium Severity Findings

[M-01] Protocol does not support rebase or fee-on-transfer tokens

Severity: *Medium risk* (*Acknowledged*)

Context: [-o-1]: *LiquifyBridge.sol*

Description

`LiquifyBridge` intends to support as many tokens as possible.

The current implementation, however, does not account for fee-on-transfer or rebase tokens, leading to funds being blocked in the contract.

Recommendation

The team must determine if supporting fee-on-transfer/rebase tokens is useful. It will require using `balanceOf` in each case to determine if the actual available amount of tokens is different that the accounted one. This adds a significant overhead that may not be worth the benefit.

While tokens that employ these mechanisms are usually in the meme-coin category, there are also DeFi related tokens such as Lido's rebasing token stETH.

However, these have a wrapped equivalent which does not rebase allowing for deposits/withdrawals (e.g. wstETH for Lido)

Resolution: Acknowledged by the team.

ABA: team has decided to drop support for fee on transfer or rebate tokens and only support 18 decimals, whitelisted, tokens.

3.3 Low Severity Findings

[L-01] Cannot deploy when owner is not deployer

Severity: *Low risk* *(Resolved)*

Context: [-o-1]: ***LiquifyBridge.sol****:57*

Description

The LiquifyBridge contract takes an _owner constructor parameter that is set using OpenZeppelin's Ownable implementation.

The constructor then calls setTrustedSigner, a function which has the onlyOwner modifier.

This leads to the contract deployment failing if msg.sender != _owner.

Recommendation

Either set the trusted signer directly in the constructor bypassing the setTrustedSigner function (a private _setTrustedSigner can be implemented, without the modifier, which is called both from the constructor and from the public version) or pass the msg.sender to the Ownable constructor.

Resolution: Resolved, the recommended fix was implemented in **#32f4614**.

[L-02] Setup signatures don't expire

Severity: *Low risk* *(Acknowledged)*

Context: [-o-2]: ***LiquifyUnifiedBridge.sol****:182-206*

Description

When a user configures a project for deposit on the LiquifyUnifiedBridge, alongside the input he passes in the call, he also receives a signature, signed by the protocol to validate the input data.

Within this signature, which validates an EIP712 signing schema, there is a lack of an expiration date (similar to how claiming has an expiration date).

Without an expiration date, a user can postpone his project launch indefinitely, for various reasons.

Recommendation

Add an expiration deadline for the EIP712 Setup signature.

Resolution: Acknowledged by the team.

[L-03] Dust is lost on deposits due to rounding

Severity: *Low risk* *(Acknowledged)*

Context: [-o-2] : ***LiquifyUnifiedBridge.sol****:234-249*

Description

When a project is added to the bridge contract, a list of percentages that represent the amounts that will be released after each vesting period, is passed.

The amount per slice is calculated by using the provided vesting period as BPS percentages out of the total amount.

The division by `MAX_BPS` when calculating the amount, will leave out dust tokens on each slice calculation. This dust should normally be included in the required deposit amounts and subsequently be received by users.

This will also result in `ProjectState::depositedAmount` not being equal to `ProjectState::totalSupply` but by a few WEI when the issue manifests.

Recommendation

Track exactly how many tokens the slices generated and using this value, determine and add any left-over dust to one of the vesting periods.

Resolution: Acknowledged by the team.

3.4 Informational Findings

[I-01] Event parameter inconsistencies

Severity: *Informational* *(Resolved)*

Context: [1] : ***LiquifyBridge.sol****:54-55*

Description

The `EmergencyWithdraw` event's first parameter is named `owner` but the emitted value is the `initiator`.

A similar issue is present for the `ApproveOwnerToClaim` event, where the first parameter name is `owner` but the emitted value is `msg.sender`.

These inconsistencies may lead to confusion or incorrect log interpretation.

Recommendation

For both events, update either the parameter names or the emitted values, to match the expected behavior.

Resolution: Resolved. A fix was implemented in **#e01a2de**.

ABA: the emergency withdraw functionality has been removed and with it, the emission of the EmergencyWithdraw event.

[I-02] Storage handling ca be simplified

Severity: *Informational* *(Resolved)*

Context: [1] : ***LiquifyBridge.sol***

Description

Throughout the codebase there are instances where the code that handles storage can be simplified. Whenever the state of the token allocation is retrieved, multiple `initiatorTokenAllocationState[<address>][<address>]` calls are made.

To simplify the code, consider adding using an intermediary variable.

Recommendation

Use intermediary storage variables where possible. Example:

```
InitiatorTokenAllocationState storage tokenAllocation =
    initiatorTokenAllocationState[msg.sender][token];

if (tokenAllocation.depositedAmount
     - tokenAllocation.claimedAmountByUsers
     - tokenAllocation.approvedAmountToBeClaimedByOwner < amount)
revert InsufficientDepositedAmount();
```

Resolution: Resolved. A fix was implemented in **#e01a2de**.

ABA: finding was resolved by removing and rewriting the entire logic.

[I-03] Remove debug remnants

> **Severity:** *Informational* *(Acknowledged)*
>
> **Context:** [-o-1]: ***LiquifyBridge.sol***:*14,187-226*

Description

Throughout the codebase there are instances where the debug code was left:

- the `// Add these custom errors at contract level` comment near the custom errors
- the `verifySignature` test function and section

Recommendation

Remove the indicated code.

Resolution: Acknowledged by the team.

[I-04] Missing events on significant action

> **Severity:** *Informational* *(Resolved)*
>
> **Context:** [-o-1]: ***LiquifyBridge.sol***:*173-176*

Description

Events help off-chain monitoring systems and are usually beneficial to have on each major smart contract action.

The current implementation, however, does not emit events on all significant actions.

Recommendation

Emit an event with relevant information when calling the `setTrustedSigner` function

Resolution: Resolved. A fix was implemented in **#6ec39d8**.

ABA: Liquify Ventures changed the setTrustedSigner implementation to a 2-step model and added proper event emissions.

[I-05] Use a two-step ownership transfer routine

> **Severity:** *Informational* *(Resolved)*

> **Context:** [-o-1] : ***LiquifyBridge.sol***

Description

When transferring ownership of a contract, mistakes that transfer the ownership to an unwarned address can be avoided by using a two-step transfer routine.

In the first step a new owner is proposed and in the second step the new owner must accept the ownership.

Recommendation

Use OpenZeppelin's 2-step ownership transfer contract in the `LiquifyBridge` contract

Resolution: Resolved, the recommended fix was implemented in **#dc17890**.

[I-06] Add security contact to contracts

> **Severity:** *Informational* (*Resolved*)
>
> **Context:** [-o-1] : ***LiquifyBridge.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 contracts.

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 contracts.

Resolution: Resolved, the recommended fix was implemented in **#09a2f2b**.

[I-07] getVestingReleaseAmounts and getProjectVestingAmounts are identical

> **Severity:** *Informational* (*Resolved*)
>
> **Context:** [-o-2] : ***LiquifyUnifiedBridge.sol***:*411-416*

Description

There are two helper view functions in the `LiquifyUnifiedBridge` contract, `getVestingReleaseAmounts` and `getProjectVestingAmounts` which are identical in behavior and both return the `vestingReleaseAmounts::vestingReleaseAmounts`.

Having 2 identical functions is redundant.

Recommendation

Remove the `getProjectVestingAmounts` function.

Resolution: Resolved, the recommended fix was implemented in **#3e8cef5**.

[I-08] Nonstandard _key view function name

> **Severity:** *Informational* *(Resolved)*
>
> **Context:** [⊸2]: ***LiquifyUnifiedBridge.sol****:388-396*

Description

In the `LiquifyUnifiedBridge` contract, there is a `_key` public function, which forms a key from the input to the `claimed` mapping.

The name of this function does not comply with the official Solidity style guide and by using the _ for a non-private/internal function.

This is both a deviation from proper coding standards and may confuse external integrators.

Recommendation

Rename the `_key` function to a more suggestive name, such as `formClaimedKey`.

Resolution: Resolved, the recommended fix was implemented in **#586bd06**.

[I-09] Bridge contract can be simplified or optimized

> **Severity:** *Informational* *(Partially Resolved)*
>
> **Context:** [⊸2]: ***LiquifyUnifiedBridge.sol****:216-222,225,280,283-284,288,300*

Description

In the `LiquifyUnifiedBridge` contract, there are several operations that can be optimized or simplified:

In the `_configProject` function, there are several local variables declared, which then are used only once:

```
uint256 raised = config.raisedAmount;
uint256 price = config.pricePerToken;
uint256 ratioNum = config.ratioNumer;
uint256 ratioDen = config.ratioDenom;
uint16[] memory pcts = config.vestingReleasePercentages;
address tokenAddr = config.tokenAddress;
```

These can be removed and instead directly use the `config.*` variable.

In the same function, the `1e18` hardcoded value can be saved in a constant for better clarity and with a suggestive name, such as `TOKEN_DECIMALS`.

In the `_deposit` function, the `st.vestingReleaseAmounts[idx]` variable is read 5 times from storage. Save this into a local variable and reuse it.

Recommendation

Apply the mentioned changes.

Resolution: Partially resolved by the team in **#be46b2f**.

ABA: only the last suggestion, to have a local variable for the multiple-times-read storage variable, has been implemented.

[I-10] Cleanup codebase

> **Severity:** *Informational* *(Partially Resolved)*
>
> **Context:** `[-o-2]:` *ILiquify.sol ILiquifyBridge.sol LiquifyBridge.sol LiquifyUnifiedBridge.sol*:370

Description

The codebase has several development leftovers that should be removed.

- unused or older versions of the contract should be removed: `LiquifyBridge.sol`, `ILiquifyBridge.sol` and `ILiquify.sol`.
- in the `LiquifyUnifiedBridge` contract, the comment on L370 is a leftover from previous deployments: `// a) global claimed mapping to prevent double-claims per project/user` and, as it is, is misleading. Remove it.
- unused errors in the `ILiquifyUnifiedBridge` interface: `InsufficientApprovedAmount`, `InsufficientDepositedAmount`, `ContractPaused`, `InvalidToken`, and `ZeroDepositAmount`. Remove them.

Recommendation

Implement the mentioned changes.

Resolution: Partially resolved by the team in **#9ef6e66**.

ABA: the incorrect comment and unused errors part was resolved.

[I-11] Missing validation for 18-decimal token assumption

> **Severity:** *Informational* *(Acknowledged)*
>
> **Context:** `[-o-2]:` *LiquifyUnifiedBridge.sol*:225

Description

The protocol intends to only support 18 decimal tokens, but there is no validation to ensure that the token specified during project configuration actually uses 18 decimals.

If this check is not performed off-chain before generating the signature for `configProjectAnd-Deposit`, the resulting `totalSupply` calculation may be incorrect, leading to inaccurate token distributions.

Recommendation

Add a check in the `_configProject` function to ensure that the token's decimals value is 18, or modify the `totalSupply` calculation to support tokens with varying decimal configurations.

Resolution: Acknowledged by the team.

4 Disclaimer