Code Assessment
of the Uniswap Liquidity Amplifier Smart Contracts

PUBLIC

Date July 30, 2026

Produced for Frankencoin

By CHAINSECURITY

Contents

1 Executive Summary

Dear Frankencoin Team,

Thank you for trusting us to help Frankencoin with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Uniswap Liquidity Amplifier according to **Scope** to support you in forming an opinion on their security risks.

Frankencoin implements Uniswap Liquidity Amplifier, which allows creating leveraged Uniswap V3 LP positions for ZCHF, using the position itself as the collateral.

The most critical subjects covered in our audit are asset solvency, functional correctness, and precision of arithmetic operations. Arithmetic precision is improvable, as rounding directions currently do not favor the protocol, see **Amounts Owed by the User Round Down**.

Security regarding the other subjects is high.

The general subjects covered are correctness of the Uniswap V3 integration and correctness of comments. Multiple comments were found to be stale, see **Outdated and Inaccurate Comments**.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,
ChainSecurity

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Uniswap Liquidity Amplifier repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

Version	Date	Commit Hash	Note
1	27 July 2026	a2cff1152a465328a6daec2a01a6dfa672ae23d1	
2	30 July 2026	c9f18c6708c699617d11324e1416f11ad4079fa6	Fixes

For the Solidity smart contracts, the compiler version `0.8.24` was chosen.

```
contracts/swap/
└── UniswapAmplifier.sol
```

2.1.1 Excluded from scope

Anything not explicitly listed above is out of scope.

3 System Overview

This system overview describes the initially received version (**Version 1**) of the contracts as defined in the Assessment Overview.

Frankencoin offers ZCHF, a Swiss Franc stablecoin backed by collateral. Uniswap Liquidity Amplifier lets users open leveraged Uniswap V3 liquidity positions in one hardcoded ZCHF pool. The ZCHF side of the position is freshly minted by the Frankencoin protocol. The other side, a dollar token, is supplied by the position owner. The combination of the ZCHF and the dollar token in the Uniswap LP position is treated as the collateral for the minted ZCHF. At the minimum allowed collateral the owner provides 41% to 47% of the position's notional value, so fee income per unit of own capital is amplified by 2.13x to 2.45x. The owner earns all of the position's swap fees and owes the minted ZCHF back to the protocol.

`UniswapAmplifier` is a factory and debt ledger. `AmplifiedPosition` is the per-user contract it deploys. Each position acts as the Uniswap V3 position owner on behalf of the user.

3.1 Amplifier parameters

The amplifier is immutable. All parameters are constants or set at construction time. There are no setters.

- **Price anchor.** The pool price at deployment, expressed as dollars per ZCHF.
- **Tick band.** Positions may only be opened in a range around the anchor tick that permits the dollar to weaken by about 14% and to strengthen by about 10%.
- **Collateral factor.** A position must be paired with dollars worth at least 80% of its borrowed ZCHF valued at the anchor, so the debt can be at most 1.25 times the dollars posted (price at the anchor) or at most 1.37 times the dollars posted (priced at the upper end of the hardcoded price range).
- **Expiration.** After expiration, no new ZCHF can be borrowed and anybody can close positions.
- **Borrowing limit.** Caps the ZCHF outstanding across all positions of the amplifier at any point in time. Repayments free up capacity.

`exploitableAt()` reports the dollar price below which the amplifier becomes exploitable, derived from these parameters. With the parameters of **Version 1** it sits about 41% below the anchor. This means that if the dollar token (expected to be USDT) falls more than 41% against ZCHF relative to the anchor price before expiration, the Amplifier can be left with bad debt.

3.2 Opening and closing a position

Anyone may call `createAmplifiedPosition()` with a tick range, which deploys a position owned by the caller. The range must lie inside the band and is fixed for the lifetime of the position.

The owner adds liquidity with `mint()`, choosing a liquidity amount. The amplifier pulls the required dollars from the owner, which needs an ERC-20 approval, mints the required ZCHF, and sends both to the pool. How much of the position must be funded with dollars follows from where the live price sits inside the range, and the call reverts if the position would borrow ZCHF against too few dollars.

The owner can remove liquidity with `burn()`. The withdrawn principal and all fees accrued so far are sent to the caller, and a proportional share of the debt is repaid by the caller. In case more ZCHF are needed to repay the debt than the LP returns, the rest is taken from the caller's balance.

Both `mint()` and `burn()` take an expected pool price for slippage protection and revert if the price deviates from it by more than roughly 0.1%. Passing zero disables that guard.

3.3 Expiration and wind-down

After expiration, anyone can repay any position using `expiredPublicBurn()`. Position owners are incentivized to close their position before expiration. Otherwise, they lose their principal. As long as the dollar price has not declined below `exploitableAt()`, it is profitable to repay someone else's position. Below that price it may be unprofitable (depending on what range the position was created at) and positions may remain open without repayment.

3.4 Changes in Version 2

In **Version 2** AmplifiedPositions are now deployed as EIP-1167 immutable proxies to save on deployment gas.

4 Trust Model

4.1 Roles

Position Owner

- Trust level: Untrusted.
- Permissions: Open a leveraged position, but only within the allowed tick range and with sufficient USD collateral.

4.2 External dependencies

Frankencoin Governance

Frankencoin governance is expected to check that the Amplifier was correctly configured before allowing it as a minter. It is also expected to cover bad debt in case this is needed, see Governance Needs to Cover Bad Debt in the Worst Case.

Tokens

The USD token is expected to be a standard ERC-20 token without special features such as rebasing, transfer hooks (reentrancy), fee-on-transfer, etc.

The USD token's value is expected not to decline against ZCHF below the threshold reported by `exploitableAt()` before the Amplifier expires.

5 System Considerations

We leverage this section to highlight findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require a modification inside the project. Instead, they should raise awareness in order to improve the overall understanding for users and developers.

SC1 Governance Must Verify Deployment Parameters

System Consideration	Version 1

Before accepting UniswapAmplifier as a minter, Frankencoin governance must verify that:

- `UNISWAP_POOL` is the intended ZCHF/dollar pool and was created by the canonical Uniswap V3 factory.
- `PRICE_ANCHOR_X96` matches the unmanipulated market price at deployment.
- `LIMIT` caps the mintable ZCHF at an acceptable amount.
- `EXPIRATION` is sufficiently near in the future that the USD coin's depreciation over that time can be estimated.
- `ZCHF` and `ZCHF_MINTER` both point at the real Frankencoin token. They are independent constructor arguments and the contract checks neither.
- The minter rights are granted to the amplifier itself, not to an `AmplifiedPosition`. The ZCHF leg of `borrowIntoPool()` is a plain `transferFrom` from the owner and relies on the implicit infinite allowance that Frankencoin grants to registered minters.

SC2 Governance Needs to Cover Bad Debt in the Worst Case

System Consideration	Version 1

In UniswapAmplifier, an owner borrows newly minted ZCHF against USDT collateral, and both sides are deposited into the Uniswap pool.

If the USD coin (USDT in the initial deployment) falls against ZCHF below the price returned by `exploitableAt()`, there is no longer an incentive to repay the ZCHF. This includes the cases where USD coin depegs downwards compared to the dollar or ZCHF depegs upward compared to the Swiss Franc. This is part of the intended model.

If this happens (which is assumed to be unlikely), the contract does not have an explicit way to handle it. For comparison, the MintingHub has an explicit mechanism that allows Frankencoin Equity to cover bad debt. As a result, Frankencoin governance (or another altruistic party) would need to become active to repay/burn any outstanding ZCHF that is no longer sufficiently backed.

6 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- **Likelihood** represents the likelihood of a finding to be triggered or exploited in practice
- **Impact** specifies the technical and business-related consequences of a finding
- **Severity** is derived based on the likelihood and the impact

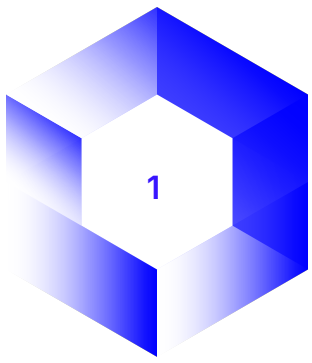
We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	**Impact**		
	High	Medium	Low
High	**Critical**	**High**	**Medium**
Medium	**High**	**Medium**	**Low**
Low	**Medium**	**Low**	**Low**

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

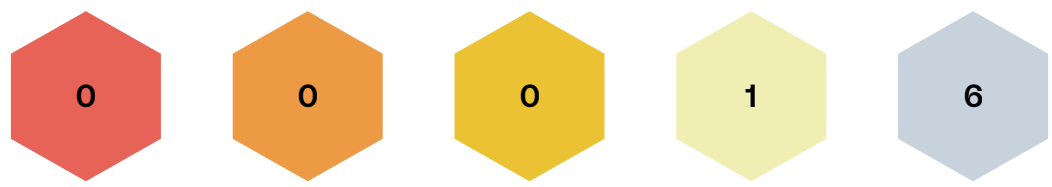
7 Findings

In this section, we describe the findings identified during the course of the engagement. The findings are categorized by severity as explained in the Terminology section. Below we provide an overview of the total number of findings per severity, as well as the number of open issues.



Open Issues

Total number of findings



Critical High Medium Low Informational

7.1 Overview

The following table lists all identified findings along with their severity and current status. A finding is considered resolved once it has been fully corrected or the specification has been changed accordingly. Non-informational findings with any other status, including partial fixes, acknowledgements, and accepted risks, remain open.

Low	#001	Amounts Owed by the User Round Down	Risk Accepted	ⓘ
Info	#002	`exploitableAt()` Can Slightly Understate the Price at Which the Amplifier Becomes Exploitable	Acknowledged	⊖
Info	#003	`token1` Is Not Validated	Code Corrected	✓
Info	#004	Collecting a Position's Accrued Fees After Expiration Is Free	Acknowledged	⊖
Info	#005	Comment Misstates Slippage Check	Code Corrected	✓
Info	#006	Contract Creation Will Become Expensive	Code Corrected	✓
Info	#007	Outdated and Inaccurate Comments	Code Corrected	✓

7.2 Descriptions

#001 Amounts Owed by the User Round Down

Correctness	Low	Version 1	Risk Accepted

In UniswapAmplifier, amounts the user owes are rounded down, so the user can owe slightly less than the exact value, or nothing at all.

1. `getMinimumDollars()` returns `Math.mulDiv(PRICE_ANCHOR_X96, zchfAmount, Q96) * 4 / 5`, which is zero whenever `PRICE_ANCHOR_X96 * zchfAmount < 2 * Q96`. For a dollar stablecoin with 6 decimals at 1.22 USD/CHF, `PRICE_ANCHOR_X96` is about $1.22 \cdot 10^{-12} \cdot 2^{96}$, so any borrow up to about $1.64 \cdot 10^{12}$ wei ZCHF (1.64 millionths of a Franc) requires no dollars at all. This is limited to griefing, because the borrowed ZCHF ends up in the pool rather than with the user and any indirect benefit is dwarfed by gas costs.

2. `repay()` burns `Math.mulDiv(borrowed, returnedPart, total)` from the user, understating the debt by up to one wei ZCHF per call.

While the impact here is limited, rounding in favor of the protocol instead of users is considered a best practice that reduces the likelihood of a whole class of exploits.

> **Risk Accepted**
> Frankencoin has chosen not to make a change in response to this finding.

#002 `exploitableAt()` Can Slightly Understate the Price at Which the Amplifier Becomes Exploitable

Informational	Version 1	Acknowledged

In UniswapAmplifier, the constructor takes two readings of the same pool state. `PRICE_ANCHOR_X96` comes from the pool's exact (sub-tick) price. `MINIMUM_TICK`, which bounds the range every position must sit inside, comes from the pool's current tick, which is that same price rounded down to a whole tick. The two agree only when the price sits exactly on a tick boundary. `exploitableAt()` combines both readings, so the price it treats as the bottom of the allowed range is off by the distance the pool price sat into its tick at the time of deployment.

The function always reports that the amplifier holds until the dollar has declined **41.3435%** below the anchor. The real threshold depends on the sub-tick price anchor. For a pool with a tick spacing of 1 (which the currently most liquid ZCHF/USDT pool uses), single-tick positions are allowed, and that is the case where the two readings diverge most:

pool price at deployment	real threshold
on a tick boundary	41.3458%
50% through its tick	41.3443%
74% through its tick	41.3435%
95% through its tick	41.3429%

Below roughly 74% the reported threshold is conservative. Above that point, the amplifier is already exploitable before the reported price. At any tick spacing wider than 1 the reported threshold would be conservative at every sub-tick price. The difference is small enough that it is negligible in practice.

Acknowledged

Frankencoin acknowledges the issue and has decided not to make a change.

#003 `token1` Is Not Validated

Informational	Version 1	Code Corrected

In the `else` branch of the constructor, it is assumed that `UNISWAP_POOL.token1() == zchf_`. The code never checks this. It could be checked to catch a deployment mistake.

Code Corrected

The `token1` is now checked to be ZCHF.

#004 Collecting a Position's Accrued Fees After Expiration Is Free

Informational	Version 1	Acknowledged

In UniswapAmplifier, `expiredPublicBurn()` lets anyone wind down a position once the amplifier has expired. It forwards to `_burn()`, which withdraws the full uncollected balance of the position, both principal and accrued swap fees, to the caller. It repays a share of the debt proportional to the liquidity burned.

Passing a burn amount of zero liquidity is accepted. The caller then receives all fees the position has accrued while repaying nothing, so the fees earned by the position owner can be taken by any address after expiration.

This behavior is in line with the fact that before expiration a position's fees are also paid to the caller, who is then necessarily the owner.

If repaying the position is profitable, the position is still expected to be repaid in full even after the fees have been collected by someone else.

Acknowledged

Frankencoin acknowledges the issue and has decided not to make a change.

#005 Comment Misstates Slippage Check

Informational	Version 1	Code Corrected

`checkPrice()` checks the upper bound as `(price * 999) / 1000 > expectedPriceX96`, which allows the price to be up to $\frac{1}{999}$ above the expected price, instead of the $\frac{1}{1000}$ (0.1%) the comment states. The bound the comment states would be checked if `expectedPriceX96` were multiplied instead of `price`.

The lower bound check `(price * 1001) / 1000 < expectedPriceX96` limits the deviation to $\frac{1}{1001}$ instead of the comment's $\frac{1}{1000}$.

Code Corrected

The comment was updated to correctly state the comparison mechanism.

#006 Contract Creation Will Become Expensive

Informational	Version 1	Code Corrected

`createAmplifiedPosition()` deploys `AmplifiedPosition` as a new contract every time. The proxy pattern is not used.

EIP-8037, which will soon be activated on Mainnet as part of the Glamsterdam update, increases the cost of deploying contracts from 200 regular gas to 1530 state gas per byte (state gas costs the same as regular gas). Given that `AmplifiedPosition` is almost 4000 bytes long, the cost of `createAmplifiedPosition()` could rise to about 6M gas.

While the system remains perfectly functional after the update, the gas cost borne by users may be a concern. It should be estimated and compared to the expected profit of a realistically-sized position. The cost could be reduced by using a proxy pattern.

> **Code Corrected**
>
> In **Version 2**, the system deploys `AmplifiedPosition` as EIP-1167 immutable proxies to minimize state footprint.

#007 Outdated and Inaccurate Comments

Informational	Version 1	Code Corrected

In UniswapAmplifier, `getMinimumDollars()` requires dollar collateral worth 4/5 of the borrowed ZCHF valued at the price anchor. The first four comments below seem to assume a factor of 1/2 instead of 4/5.

1. The header states that "only half of its value required as dollar collateral". The requirement is four fifths.
2. The header concludes that the owner "provides about a third of the position's value, roughly tripling the fee income per unit of own capital". The owner must supply at least 41% to 47% of the position value, depending on where the mint price sits in the allowed band, capping amplification at 2.13x to 2.45x.
3. The documentation of `borrowIntoPool()` states that borrowing 85 CHF at an initial price of 0.85 CHF/USD "also requires at least 50 USD". It requires 80 USD.
4. `expiredPublicBurn()` states that a third party can wind a position down profitably "as long as the dollar has not declined more than about 24% to 33% below the anchor price". Such a caller receives the position's tokens and burns the proportional share of the debt, so its break-even is governed by the same inequality as the owner's decision to walk away. The real range is 41% to 49% for positions posting the minimum collateral.
5. `createAmplifiedPosition()` documents both of its tick parameters as needing to be "within +/- 20% of the initial price". This describes a previous version's band of `anchorTick +/- 2000` ticks. The enforced band is `[anchorTick - 1000, anchorTick + 1500]`, which corresponds to -9.52% to +16.18% in the price the ticks encode, i.e., dollars per ZCHF. All percentages in items 5 and 6 use that convention. Expressed as a dollar price instead, the same band is -13.93% to +10.52%, which is what the header's "(actually -13.93%/+10.52%)" states.
6. The header describes buying the minted ZCHF "at the lowest allowed price (anchor -10.52%)". `MINIMUM_TICK` sits 1000 ticks below the anchor tick, and 1000 ticks down is a price 9.52% below the anchor. The 10.52% is the same 1000 ticks read upwards, i.e., the dollar appreciation the header quotes as "+10.52%" for the opposite direction. A tick move down and the same move up are not mirror images in percent terms.

7. The `exploitableAt()` documentation says winding down after expiration would happen "with the missing ZCHF coming out of the equity pool". No such path exists. Frankencoin's `coverLoss()` is neither called by the amplifier nor declared in its `IFrankencoinMinter` interface, so any shortfall has to be covered manually.

Code Corrected

1, 2, and 3 were addressed by removing the sentence. 4 and 5 were addressed by replacing numbers with references to other comments. 6 was addressed by correcting the sentence. 7 was addressed by clarifying the intended mechanism.

8 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.