ANASTASIA LABS

Security Audit Report

Surf Lending

Date 5th November, 2025
Project Surf Lending Contracts
Version 1.0

Contents

Disclosure

Nonetheless, both the customer **Surf** and Anastasia Labs are authorized to share this document with the public to demonstrate security compliance and transparency regarding the outcomes of the Protocol.

Disclaimer and Scope

A code review represents a snapshot in time, and the findings and recommendations presented in this report reflect the information gathered during the assessment period. It is important to note that any modifications made outside of this timeframe will not be captured in this report.

While diligent efforts have been made to uncover potential vulnerabilities, it is essential to recognize that this assessment may not uncover all potential security issues in the protocol.

It is imperative to understand that the findings and recommendations provided in this audit report should not be construed as investment advice.

Furthermore, it is strongly recommended that projects consider undergoing multiple independent audits and/or participating in bug bounty programs to increase their protocol security.

Please be aware that the scope of this security audit does not extend to the compiler layer, such as the UPLC code generated by the compiler or any areas beyond the audited code.

The scope of the audit did not include additional creation of unit testing or property-based testing of the contracts.

Assessment overview

From **13th August**, **2025** to **5th November**, **2025**, **Surf** engaged Anastasia Labs to evaluate and conduct a security assessment of its **Lending** protocol. All code revision was performed following industry best practices.

Phases of code auditing activities include the following:

- **Planning** – Customer goals are gathered.
- **Discovery** – Perform code review to identify potential vulnerabilities, weak areas, and exploits.
- **Attack** – Confirm potential vulnerabilities through testing and perform additional discovery upon new access.
- **Reporting** – Document all found vulnerabilities.

The engineering team has also conducted a comprehensive review of protocol optimization strategies.

Each issue was logged and labeled with its corresponding severity level, making it easier for our audit team to manage and tackle each vulnerability.

Assessment components

Manual revision

Our manual code auditing is focused on a wide range of attack vectors, including but not limited to:

- UTXO Value Size Spam (Token Dust Attack)
- Large Datum or Unbounded Protocol Datum
- EUTXO Concurrency DoS
- Unauthorized Data modification
- Multisig PK Attack
- Infinite Mint
- Incorrect Parameterized Scripts
- Other Redeemer
- Other Token Name
- Arbitrary UTXO Datum
- Unbounded protocol value
- Foreign UTXO tokens
- Double or Multiple satisfaction
- Locked Ada
- Locked non Ada values
- Missing UTXO authentication
- UTXO contention

Executive summary

Surf Lending is an AMM-style pooled-lending protocol, where users can put in ADA or native tokens as collateral and borrow other assets (specific to each pool).

Each pool has an associated "fToken," which depending on how much deposits a pool has received versus how much of these fTokens it has minted/burnt so far, liquidity providers (LPs) receive back a portion of their deposits as newly minted fTokens.

Borrowers, on the other hand, will have to pay an interest on their loans (which its rate is calculated based on the current "utilization" of a pool) that will contribute to increasing the ratio governing the value of fTokens.

It is of note that this is a centralized protocol. Core features such as token prices, available assets for borrowing, and allowed collateral assets are controled by Surf. Additionally, all the funds locked under any of the contracts can be spent by them, and also all the tokens associated with this protocol can be arbitrarily minted or burnt with a single admin signature.

Code base

Repository

`https://github.com/flow-lending/flow-lending-smart-contracts.git`

Commit

`110523b12bb06b0397a1dfb48365584766e13e9b`

Files Audited

Files | SHA256 Checksum
validators/ batcher_mp.ak `431e2a04930a7a30d99ad482f99edecdbd355b8dbfce4907d242e495545201d2`
validators/ ctoken.ak `47b091db06e7f9c606daac849df8d8603f1108267a24d67ec75baf44ceb79935`
validators/ order.ak `028f34b3c5f4dce9ecbb4355487063f3ba9122f14eb14429a367d9580a23b2c6`
validators/ pool.ak `c55713f6742916e7e2617d79dcc729c6948935cb071428d43e5fdc33b53c77f8`
validators/pool_auth_token.ak `3cb0c254db91ca4bee2f134da43fcbf2643173ea47cc2f891a41d1e7204f15e2`
validators/ pool_info.ak `edb3101b64303eee2151b9e5641621faf8511cfeb65fefab4a88fabc587616a6`
validators/ vault.ak `391c867886a322f1b505626f3ce78c69a6ddb71b8d8b991722b4e3c8f8fcff1b`
validators/vault_authtoken.ak `99f3650f894698fa7e4c6396e75ddd35c68e8b76854cb0a3ac9e868d5ae5e234`
lib/constants.ak `4676beac65459ee878117f4d612ea3e804a4872c20b6795427be576dbc0728da`
lib/types.ak `30cad34df778004110bfd3b6330457a68503f2d97320fae4bafe60c54c83c39f`
lib/utils.ak `8b8f1dae467920c4237ae20f9d9e2c5e5c082c9e4caba3fe123c2ed2e73e6891`

Severity Classification

- **Critical**: This vulnerability has the potential to result in significant financial losses to the protocol. They often enable attackers to directly steal assets from contracts or users, or permanently lock funds within the contract.
- **Major**: Can lead to damage to the user or protocol, although the impact may be restricted to specific functionalities or temporal control. Attackers exploiting major vulnerabilities may cause harm or disrupt certain aspects of the protocol.
- **Medium**: May not directly result in financial losses, but they can temporarily impair the protocol's functionality. Examples include susceptibility to front-running attacks, which can undermine the integrity of transactions.
- **Minor**: Minor vulnerabilities do not typically result in financial losses or significant harm to users or the protocol. The attack vector may be inconsequential or the attacker's incentive to exploit it may be minimal.
- **Informational**: These findings do not pose immediate financial risks. These may include protocol optimizations, code style recommendations, alignment with naming conventions, overall contract design suggestions, and documentation discrepancies between the code and protocol specifications.

Finding severity ratings

The following table defines levels of severity and score range that are used throughout the document to assess vulnerability and risk impact

Level	Severity	Status
5	Critical	0
4	Major	1
3	Medium	2
2	Minor	2
1	Informational	3

Findings

ID-401 Repayment Interest Flexibility

Level	Severity	Status
4	Major	Acknowledged

Description

The pool's repayment endpoint uses the upper bound of the transaction's validity range for calculating the accrued interest. This gives batchers a wide range to choose repayment amounts for these orders, allowing them to manipulate prices of the pools' tokens to the upside.

This flexibility of interest is also present in the liquidation endpoint.

Recommendation

Restrict the time ranges in these two endpoints to a maximum window size to limit this flexibility, which consequently ensures the upper bound can't lie too far in the future.

Resolution

Acknowledged

ID-301 Negative Collateral Amount

Level	Severity	Status
3	Medium	Acknowledged

Description

Application of borrow orders does not impose any restrictions for the collateral amounts to be positive. Meaning, if an order specifies the collateral amount to be negative, the resulting LTV will be negative, allowing borrowers to circumvent the max borrow LTV requirement.

Recommendation

Similar to how principal is checked for being positive, restrict collateral amounts to be positive as well.

Resolution

Acknowledged

ID-302 Single Point of Failure for Authorized Endpoints

Level	Severity	Status
3	Medium	Acknowledged

Description

There are various endpoints among the validators that require admin's permession (as a sole requirement or in addition to other stipulations). This, however, is implemented as the presence of a single signature. Essentially introducing a single point of failure for the protocol to function.

Recommendation

Either use a multisig for admin's permissions, or define a specific NFT that its expenditure provides the permission.

Resolution

Acknowledged

ID-201 Incorrect Relation Between Max Borrow LTV and Liquidation Threshold

	Level	Severity	Status
	2	Minor	Acknowledged

Description

Collaterals of a new pool info are only checked to ensure both max borrow LTV and liquidation threshold are above zero (defined in the `is_pool_info_datum_valid` helper function). However, this doesn't prevent the case of max borrow LTV being higher than the liquidation threshold. Any successful borrow from such a pool would immediately be subject to liquidation.

Recommendation

The validation should be updated to ensure the threshold is larger than LTV, and possibly with a hardcoded buffer.

Resolution

Acknowledged

ID-202 Unvalidated Asset Names for Batcher Tokens

	Level	Severity	Status
	2	Minor	Acknowledged

Description

At minting batcher tokens, deadlines of licenses are currently provided via the redeemers, but this doesn't add any additional restrictions as the asset names themselves are indicated in the transaction.

Additionally, there are no validations to make sure the asset names can later be decoded into POSIX time strings (which is a requirement for all pool endpoints). This means batchers need to rely on the integrity of admin's off-chain code for the correctness of their license NFTs.

Recommendation

Remove the provision of deadline bytearray from the redeemer and instead simply grab it from the mint field, decode it into an `Int` and make sure that it is set for some time in the future.

Resolution

Acknowledged

ID-101 Missing Impact of Orders on Pool Utilization in Each Batch

	Level	Severity	Status
	1	Informational	Acknowledged

Description

The current fold logic under pool endpoints does not take into account the impact of the orders it traverses on the pool, and instead uses the initial utilization of the pool from its datum.

This approach makes the relationship between utilization and interest rate closer to a step function rather than a linear one, and has the potential of unjustly subjecting more/fewer users to the rate spike at kink.

Recommendation

The accumulator of the fold function should also carry along the updated utilization of the pool so that each order is impacted by the order that comes before it.

Resolution

Acknowledged

ID-102 Random Application of Orders

	Level	Severity	Status
	1	Informational	Acknowledged

Description

Orders are currently applied in batches, however their ordering is subject to the lexicographical ordering of transaction inputs.

Aside being less fair than a First-In-First-Out (FIFO) scheme, it also allows users to "mine" transaction metadata in order to find smaller transaction hashes to control where they end up in a batch.

Switching to FIFO would also demand support of applying different order types in same batches (which is currently not the case as batches are classified by their types).

Assuming vulnerability 01 is also addressed as suggested, the ability of users to control their position in a batch allows attackers to push pending borrow orders to fall past the kink.

Recommendation

Enforce FIFO application of orders to pools. That is, instead of validating each provided order index is larger than its previous one, use the provided list of indices on a "bitmask" to prevent duplicates.

One possible implementation:

```
use aiken/builtin.{count_set_bits, replicate_byte, write_bits}
use aiken/collection/list.{length}

let bitmask = replicate_byte(0, 32)
let flipped_bits = write_bits(bitmask, order_indices, True)
expect count_set_bits(flipped_bits) == length(order_indices)
```

Aiken

Since batchers are licensed, the enforcement of upholding FIFO can be done in a centralized manner.

Resolution

Acknowledged

ID-103 Redundancies and Optimization Opportunities

	Level	Severity	Status
	1	Informational	Acknowledged

Description

1. In the minting endpoint of the pool authorization contract, both outputindices and intended destination addresses are provided via the redeemer. Since the addresses are not used anywhere else, validating their equality with the outputs' are redundant and won't add any extra restrictions.
2. `CTokenAction` of `ctoken.ak` minting endpoint delegates its logic to specific redeemers of the pool contract. Its validation of the pool UTxO is based on its state token—a check that is already performed by the pool script.
3. Same as item 2 above, but for the minting policy in `vault_authtoken.ak`.
4. The `VSpend` endpoint of the vault contract also has a similar redundancy.
5. Recalculation of input Lovelace count at line 839 in `pool.ak`.
6. Code duplication for finding the repayment amount in both the repayment and liquidation endpoints.
7. There is a recalculation of the repayment amount based on the time specified by the order UTxO (`o_time`, which has minimal validation), as the lower bound of the input amount. While, because of other restrictions in place, this doesn't seem to pose any financial risks, it reduces readability of the code, and simply consumes unnecessary resources.

Recommendation

1. Simply remove the provision of destination addresses (and consequently their equality checks) to reduce transaction fees for this endpoint.
2. By parametrizing the `ctoken` contract by pool's script hash rather than its state token, the same delegation guarantee can be achieve a bit cheaper (i.e. by ensuring the UTxO is being spent from said script hash).
3. Same as item 2.
4. Same as item 2 and 3.
5. Reuse `in_lovelace` on the left side of comparison.
6. Define a dedicated utility function to reuse.
7. Consider removing `o_time` from the repayment order's datum, and therefore the recalculation snippet.

Resolution

Acknowledged