



NOWITNESS

Security Audit Report

Flexible Rate Lending

Date August 7, 2025
Project Danogo
Version 1.2

Contents

Disclosure	1
Disclaimer and Scope	2
Assessment overview	3
Assessment components	4
Executive summary	5
Code base	6
Repository	6
Commit	6
Files Audited	6
Severity Classification	8
Finding severity ratings	9
Findings	10
ID-501 Staking Contract Drain	11
ID-502 Robbed Staking Rewards	13
ID-401 Pool DDOS	15
ID-402 TopupWithdraw Pool DDOS	17
ID-301 Incorrect alt_supply_tokens_rate list order possible	19
ID-101 Optimize Fetching Head of List	21
ID-102 Unnecessary Option Wrapping	22
ID-103 Optimize Datum Check	24
ID-104 Optimize Input Iteration	25

Disclosure

This document contains proprietary information belonging to No Witness Labs. Duplication, redistribution, or use, in whole or in part, in any form, requires explicit consent from No Witness Labs.

Nonetheless, both the customer **Danogo** and No Witness 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 **25th May, 2025** to **28th July, 2025**, **Danogo** engaged No Witness Labs to evaluate and conduct a security assessment of its **Flexible Rate 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

Flexible Rate Lending protocol is a decentralized lending and borrowing platform that allows liquidity providers to supply assets and earn interest or borrow against their collaterals. Interest rates are dynamically determined by market supply and demand that allows for real-time adjustments. With Danogo's flexible pool model, users interact directly with the protocol rather than negotiating terms such as interest rate, maturity or collateral directly with another party. Protocol also allows idle pool funds to be deposited into Liqwid to maximize capital utilization.

The protocol heavily relies on two different types of reference inputs authenticated by Protocol Config NFT and Market Param NFT to obtain `ProtocolDatum` and `MarketDatum` respectively.

- `ProtocolDatum` serves as a source of authorized script credentials referenced by protocol scripts to ensure correct script dependencies and facilitate upgrades.
- `MarketDatum` supplies critical parameters like list of tokens eligible as collateral (including token metadata like liquidity threshold), protocol fees, interest rates and fee address to individual pools. License NFT is used for authorizing updates to all `MarketDatum`s.

It is crucial to note that manipulating these datum values can result in draining or locking of protocol funds and incorrect price calculations, resulting in financial losses. Currently, these NFTs are directly managed by the Danogo team using dedicated validators. They have envisioned to transition from centralized control to a DAO based management model in the future.

Code base

Repository

<https://github.com/Danogo2023/float-rate-lending>

Commit

e86ba6616644b007ce972c2beab43ae1c19f1069

Files Audited

Files SHA256 Checksum
validators/loan.ak 081d4d8ce6deb57404bc0ed4ce5e0acea614c9cd3ba6e106a00752190810044a
validators/pool.ak e11c89fa562bbad11e500454065a2ca94b147665d166fc3e83f15fc5dfb59e25
validators/pool_config.ak 2b7b606d60d49a71f34800d81966c4dd91d6ac22fe6f621bc743c2fe71e2041b
validators/staking_contract.ak 36b2d50d9999beb0f864134b5d5e98c970080e5de880349db55ea2bd83efc7f7
lib/loan/create_loan.ak e716ad1052e80fd31cac49df537202c801b5adbd8ea875b1d8a8def325379cc2
lib/loan/decrease_loan_amount.ak 31f7217da4afd333256903d472db6ac725563dfcb0b9fbf683074c7da433bf3b
lib/loan/liquidate.ak 9008db7ab612dfd0e6cac76d8028a92356ec6ecb58f7da95f29c4922ea48bcaa
lib/loan/modify_collateral.ak f1fdf18fe7d72de190355269a6117e6928d1a94c9215e48f1f7255ba937db400
lib/loan/utills.ak 333f25ddddc629eead9a8f28880b06608e82651bbdd5462270e943b558059562






Files SHA256 Checksum
lib/pool/collect_loan.ak a8cc8291f4c73d1fa0369e6a017d76b2f148f0656c228f5f4535ec3b2df3b6a1
lib/pool/increase_loan_amount.ak eb2dba9e049ea42b17151c93d028f85f231e769d9f8dd4c29999314a257b52dd
lib/pool/rebalance_alternative_tokens.ak 6971bf5b2aa0475c190504299554a4c88a567150952d1e4efb64a91192d462cc
lib/pool/topup_withdraw.ak 01cd092f64d2a053acd4d7ed55551b917b8ab9e68b5b9ab1346376bfbceec6488
lib/pool/utls.ak ee6df949fd53b59d171f00a8593eb515148de008e318034c9b169f8fd01b9f97
lib/pool_config/create_pool.ak 54edec67a80bb486be88ed777184de6ad59f66c617f6be985f811e037e7fadb8
lib/pool_config/update_market_param.ak 64592172e5253228c10abb8c82e087d97566b2cbe0c55efef13faef93d9e832f
lib/staking_contract/create_contract.ak a827d626e29333bb0ce5e6b9ba309d22496895a01154b56f3992a7afd7c7e221
lib/staking_contract/topup_withdraw.ak 998606134dcd4c07f88a3c0b458dbfc2ad21ac622b1bd99153b3f070c54b25fa
lib/staking_contract/update_delegation.ak 57b121a1b03d74e7c809f3ff9c76192efaafb58500bd3aa89f162cd078ae5de1
lib/constants.ak c245bea396e93bfd433d72356901b32385946c6ea7bdced2dc9489cec42c613e
lib/types.ak bee24e030967fee9b812983ca50b175bf2d7aaa117481f1c3fabd032228bb255
lib/utls.ak 4959b69f70c82bcf7198dd8164fe02871f9315198be8371ddc4e1d11f08935a8

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	2
	4	Major	2
	3	Medium	1
	2	Minor	0
	1	Informational	4

Findings

ID-501 Staking Contract Drain

Level	Severity	Status
 5	Critical	Resolved

Description

When certain \$ADA amount is being withdrawn from Staking Contract, equivalent amount of \$sADA must be burnt. This however is not the case in the current implementation wherein equivalent amount of \$sADA can be minted.

```

1  // lib/staking_contract/topup_withdraw.ak Aiken
2  and {
3      if topup_withdraw_amt > 0 {
4          expect Some(expected_minted_amount) =
5              rational.div(rational.from_int(topup_withdraw_amt),
6                  output_stoken_rate)
7          (minted_amount == rational.floor(expected_minted_amount))?
8      } else {
9          (topup_withdraw_amt == -rational.floor(
10             rational.mul(
11                 rational.from_int(math.abs(minted_amount)), // <- allows positive
12                 $sADA mint amount when withdrawing $ADA (topup_withdraw_amt < 0)
13                 output_stoken_rate,
14             )),
15             (tx_end <= end_of_epoch)?,
16             (contract_in_address == contract_out_addr)?,
17             utils.contain_only_nft(contract_out_value, pid, contract_nft_name)?,
18             (contract_out_datum == InlineDatum(expected_output_datum))?,
19             (contract_out_reference_script == None)?,
20             (assets.lovelace_of(contract_out_value) - env.min_contract_ada >=
21                 output_total_supply)?,
22             or {
23                 withdrawal_amount > 0,
24                 math.abs(topup_withdraw_amt) >= env.staking_min_tx_amount,
25             }?,
26         }
27     }

```

In a single transaction any actor, irrespective of whether he is a liquidity provider or not, would be able to steal contract's entire funds.

Recommendation

To check that `minted_amount <= 0` if `topup_withdraw_amt <= 0`.

Resolution

Resolved in commit hash `7b520490d184d75e66a0ef1220514755c5e4fe12`.

ID-502 Robbed Staking Rewards

Level	Severity	Status
 5	Critical	Resolved

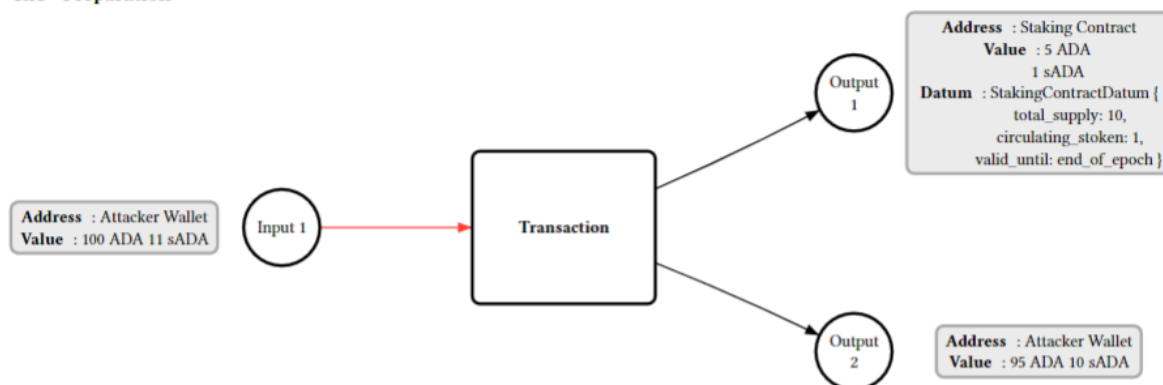
Description

Staking Contract's Withdrawal Validator delegates its validation to its own Spending Validator. While this Spending Validator intends to allow spending of just one authenticated UTxO (with Contract NFT), it fails to enforce it completely. It is possible to trick the validator into believing that \$sADA token is the Contract NFT and spend a UTxO from it satisfying Withdrawal Validator and stealing withdrawal funds.

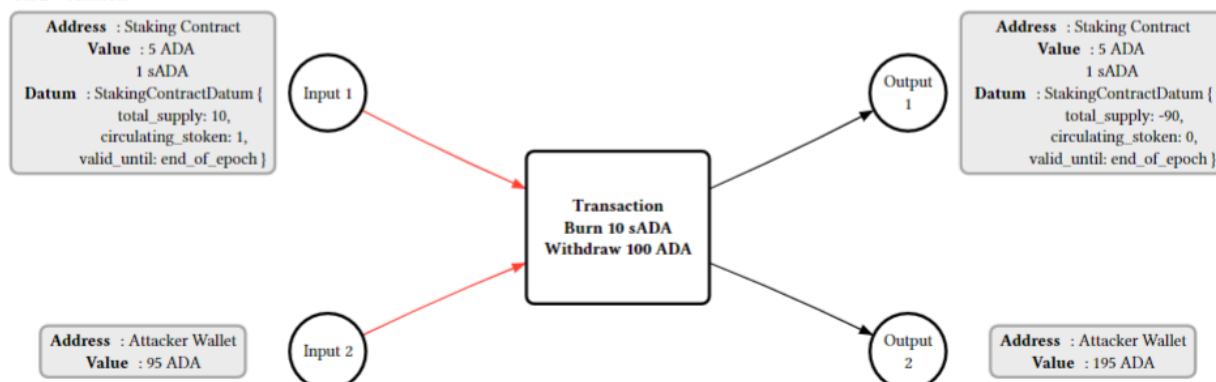
The below transaction sequence demonstrates this hack:

Robbed staking rewards transaction sequence

Tx1 - Preparation




Tx 2 - Attack



It happens since validator obtains the `contract_nft_name` from the UTxO value. Since Contract NFT and sADA both share the same Policy Id, confusion ensues.

```
1 let contract_nft_name =  
2   utils.must_get_nft_name_with_pid(contract_in_value, pid)
```

 Aiken

Recommendation

Since only two different token names could ever be minted from Staking Contract's Minting Policy checking that `contract_nft_name != ""` ensures the token is Contract NFT.

Resolution

Resolved in commit hash `7b520490d184d75e66a0ef1220514755c5e4fe12`.

ID-401 Pool DDOS

Level	Severity	Status
 4	Major	Resolved

Description

No minimum loan duration resulting in short duration loans which don't accrue any interest.

```
1  pub fn calculate_output_interest_index(✎ Aiken
2      input_interest_index: Int,
3      borrow_apy: Basis,
4      input_interest_time: Int,
5      txn_start: Int,
6  ) -> Int {
7      if txn_start > input_interest_time {
8          input_interest_index + input_interest_index * borrow_apy * (
9              txn_start - input_interest_time
10         ) / ( constants.basis_point * constants.year_in_ms )
11      } else {
12          input_interest_index
13      }
14  }
15
16  expect Some(loan_amount_with_interest) =
17      rational.new(
18          loan_in_amount * output_interest_index,
19          input_interest_index,
20  )
```

Since `output_interest_index == input_interest_index` the loan amount would not accrue any interest. A malicious actor can repeatedly spend the pool by creating and repaying such short duration loans without paying any interest to the protocol. Other protocol users would be deprived of interacting with the protocol.

Additionally, subsequent `IncreaseLoanAmount` and `DecreaseLoanAmount` actions too could result in a similar pool DDOS attack without increasing the loan amount significantly for the attacker.

Recommendation

To implement minimum loan duration and/or minimum fees to make `IncreaseLoanAmount` and `DecreaseLoanAmount` actions reasonably expensive to carry out DDOS attacks.

Note: `DecreaseLoanAmount` action is used to repay loans too.

Resolution

Resolved in commit hash `7b520490d184d75e66a0ef1220514755c5e4fe12`.

ID-402 TopupWithdraw Pool DDOS

Level	Severity	Status
 4	Major	Resolved

Description

The protocol adopts a permissionless and open design to interact with liquidity pools for addition or withdrawal of funds. One significant drawback of this is that it opens the door for DDOS attack on pool. It is even more opportunistic for an attacker compared to loan creation and repayment DDOS attack due to the following reasons:

1. Multiple pools can be spent in a single transaction.
2. Withdrawal amount from one pool can act as topup to another pool in the same transaction, reducing capital requirement imposed by `min_tx_amount` by half.
3. No specific expenses like loan origination fee or loan interest.

Recommendation

Since the protocol puts higher emphasis on complete decentralization via permissionlessness, completely mitigating the threat of DDOS becomes unachievable as some form of centralization is required. Instead the protocol can resort to certain defense mechanisms to better protect it under an attack using:

1. Rate Limiting - One way to provide an even playing field against an attacker would be to apply rate limiting to all pool spends. Let's say the limit is 1 pool spend per minute i.e. a pool cannot be spent consecutively within a minute ($in_initial_time + wait_duration < tx_start$). This would allow the protocol sufficient time to contend with attacker's transaction to spend the pool. It was unavailable before because attacker would have employed transaction chaining to continuously spend the pool without protocol obtaining the latest pool output reference. Now this would affect the protocol throughput if kept as a constant, instead `wait_duration` is kept as a configuration parameter set to zero. Should the protocol come under attack this param would be updated and the throughput would be degraded as a consequence, but only till DDOS pressure lasts possibly allowing the protocol to get some transactions through even under DDOS.
2. Permissioned Batching - It can be used as a complementary measure to rate limiting. The throughput loss due to rate limiting and DDOS can be compensated with batching should protocol be able to contend strongly to spend the pool utxo. The protocol is designed in a way that batching can be plugged in without making any changes to existing contract allowing for a hybrid proto-

col - one which is permissionless by default but resorts to permissioning via batching under DDOS pressure.


Resolution

Resolved in commit hash `7b520490d184d75e66a0ef1220514755c5e4fe12` .

ID-301 Incorrect alt_supply_tokens_rate list order possible

Level	Severity	Status
 3	Medium	Resolved

Description

1	<code>pub type MarketDatum {</code>	
2	<code>collaterals: Pairs<TupleAsset, LiquidationThreshold>,</code>	
3	<code>base_rate: Basis,</code>	
4	<code>power_base: Basis,</code>	
5	<code>util_cap: Basis,</code>	
6	<code>loan_fee_rate: Basis,</code>	
7	<code>loan_origination_fee_rate: Basis,</code>	
8	<code>fee_address: Address,</code>	
9	<code>supply_token: TupleAsset,</code>	
10	<code>alt_supply_tokens: Pairs<TupleAsset, Bool>,</code>	
11	<code>min_tx_amount: Int,</code>	
12	<code>}</code>	
13		
14	<code>pub type PoolDatum {</code>	
15	<code>total_supply: Int,</code>	
16	<code>circulating_dtoken: Int,</code>	
17	<code>total_borrow: Int,</code>	
18	<code>borrow_apy: Basis,</code>	
19	<code>undistributed_fee: Int,</code>	
20	<code>interest_index: Int,</code>	
21	<code>interest_time: Int,</code>	
22	<code>alt_supply_tokens_rate: List<PRational>,</code>	
23	<code>}</code>	

The order of input rates present in `PoolDatum`'s `alt_supply_tokens_rate` should always match the order of `MarketDatum`'s `alt_supply_tokens`. Otherwise, it would result in incorrect yield calculation of alternative tokens. There are no constraints currently in Pool Config Validator to ensure the order of already present tokens remains the same when `alt_supply_tokens` list is updated.

Recommendation

Following properties must be satisfied when updating the list:

- new tokens must always be added to the end of the list
- each token must be unique
- order of existing tokens must not be changed

Resolution


Resolved in commit hash `7b520490d184d75e66a0ef1220514755c5e4fe12`.

ID-101 Optimize Fetching Head of List

Level	Severity	Status
 1	Informational	Resolved

Description

```
1 pub fn get_nft_name_from_input(inputs: List<Input>) -> AssetName {  
2   let Input { output_reference: out_ref, .. } = flist.get(inputs, 0)  
3   outref_hash.blake2b_224(out_ref)  
4 }
```



It is more efficient to use `builtin.head_list` to fetch the first element of a list.

Recommendation

To replace `flist.get(inputs, 0)` with `builtin.head_list(inputs)`.

Resolution

Resolved in commit hash `7b520490d184d75e66a0ef1220514755c5e4fe12`.

ID-102 Unnecessary Option Wrapping

Level	Severity	Status
<div></div> 1	Informational	Resolved

Description

1	<code>pub type LendingAction {</code>	 Aiken
2		
3	<code>//</code>	
4		
5	<code> IncreaseLoanAmount {</code>	
6	<code> pool_out_idx: Int,</code>	
7	<code> loan_out_idx: Int,</code>	
8	<code> fee_out_idx: Option<Int>,</code>	
9	<code> protocol_cfg_ref_idx: Int,</code>	
10	<code> market_ref_idx: Int,</code>	
11	<code> pool_in_out_ref: OutputReference,</code>	
12	<code> }</code>	
13	<code> DecreaseLoanAmount {</code>	
14	<code> protocol_cfg_ref_idx: Int,</code>	
15	<code> maybe_withdraw_indexer: Option<DecreaseLoanAmountIndexer>,</code>	
16	<code> }</code>	
17		
18	<code>//</code>	
19		
20	<code>}</code>	
21		
22	<code>pub type DecreaseLoanAmountIndexer {</code>	
23	<code> pool_in_out_ref: OutputReference,</code>	
24	<code> maybe_loan_out_idx: Option<Int>,</code>	
25	<code> pool_out_idx: Int,</code>	
26	<code> maybe_fee_out_idx: Option<Int>,</code>	
27	<code> market_ref_idx: Int,</code>	
28	<code>}</code>	

The `None` value for optional type `maybe_withdraw_indexer` is always discarded hence it is not required to wrap the field in `Option` type.

Recommendation

To simplify the data constructor definition definition.

```
1 DecreaseLoanAmount {  
2     protocol_cfg_ref_idx: Int,  
3     pool_in_out_ref: OutputReference,  
4     maybe_loan_out_idx: Option<Int>,  
5     pool_out_idx: Int,  
6     maybe_fee_out_idx: Option<Int>,  
7     market_ref_idx: Int,  
8 }
```

Aiken

Resolution

Resolved in commit hash `7b520490d184d75e66a0ef1220514755c5e4fe12`.

ID-103 Optimize Datum Check

Level	Severity	Status
 1	Informational	Resolved

Description

```
1 // lib/pool_config/create_pool.ak 
2
3 expect InlineDatum(dt) = market_out_datum
4 expect _: MarketDatum = dt
5 expect MarketDatum {
6     base_rate,
7     power_base,
8     collaterals,
9     alt_supply_tokens,
10     ..
11 } = dt
```

Checking that datum is of type `MarketDatum` and destructuring it can be done in a single expect statement instead of two.

Recommendation

To remove the redundant `expect` type check.


Resolution

Resolved in commit hash `7b520490d184d75e66a0ef1220514755c5e4fe12`.

ID-104 Optimize Input Iteration

Level	Severity	Status
 1	Informational	Resolved

Description

1	// lib/pool/collect_loan.ak	 Aiken
2		
3	utils.foldl6(
4	inputs,	
5	None,	
6	None,	
7	assets.zero,	
8	0,	
9	0,	
10	0,	
11	fn(
12	Input {	
13	output: Output {	
14	address: Address { payment_credential: in_payment_credential, .. } as	
	in_address,	
15	value: in_value,	
16	datum: in_datum,	
17	..	
18	},	
19	..	
20	},	
21	pool_in_addr,	
22	pool_in_dt,	
23	pool_in_value,	
24	total_loan_amount,	
25	total_loan_supply_token_qty,	
26	collected_loans_count,	
27	return,	
28) {	
29	if in_payment_credential == Script(pool_skh) { // <-- True only once	
30	// check if pool market token match market param	

```
31     expect utils.contains_nft(in_value, market_token)?
32     return(
33         Some(in_address),
34         Some(in_datum),
35         in_value,
36         total_loan_amount,
37         total_loan_supply_token_qty,
38         collected_loans_count,
39     )
40 } else if in_payment_credential == Script(loan_skh) { // <-- Can be true
multiple times
41     expect InlineDatum(inline_datum) = in_datum
42     expect LoanDatum {
43         owner_nft,
44         loan_amount,
45         initial_interest_index,
46         ..
47     }
48
49     //
50 },
51 )
```

It is more efficient to list independent boolean expressions of an if-else ladder in the descending order of frequency a condition is true in a fold.

Recommendation

To place condition `in_payment_credential == Script(loan_skh)` before `in_payment_credential == Script(pool_skh)`.

Resolution

Resolved in commit hash `7b520490d184d75e66a0ef1220514755c5e4fe12`.