Tinyman AMM Contracts V1.1 - Internal Review

Pool LogicSig Template:
https://github.com/tinymanorg/tinyman-contracts-v1/blob/8be3e7f8005bb131c51e10e82885a8e764d7a336/contracts/pool_logicsig.teal.tmpl

ValidatorApp:
https://github.com/tinymanorg/tinyman-contracts-v1/blob/8be3e7f8005bb131c51e10e82885a8e764d7a336/contracts/validator_approval.teal

Transaction Specifications: https://docs.tinyman.org/integration/transactions

Actors

Pool

- A contract account controlled exclusively by a LogicSig contract generated from the Pool LogicSig Template.
- Signs all NoOp App Calls to the ValidatorApp

Pooler

- A user account that provides liquidity (2 Assets) to a Pool
- Sends asset to the Pool with a Mint operation in return for receiving Pool Tokens representing the share of the Pool's liquidity
- Sends Pool Tokens to the Pool with a Burn operation in return for receiving the correct share of the liquidity in each asset
- Claims back excess amounts of assets (due to slippage tolerance) with a Redeem operation
- The account whose local state is updated by the validator app (Accounts[1])

Swapper

- A user account that exchanges an amount of one asset for another with a Pool
- Claims back excess amounts of assets (due to slippage tolerance) with a Redeem operation
- The account whose local state is updated by the validator app (Accounts[1])

Creator

- An account that deploys the Validator App to the Algorand network

NotPool

- Any account except a Pool (may be Pooler or Swapper or other)
- Sends the Payment transaction to cover fees/min balance for Pool signed transactions.

Attack Vectors

The following is a non exhaustive list of attack vectors considered during this review.

- Send assets from pool account
    - Close account
    - Close asset
    - Include extra txns in group signed by pool logicsig
    - Change txn in valid group to do something else
    - Sign unexpected txns with pool logicsig
- Manipulate local state of a pool account
    - Increase excess amounts incorrectly
    - Redeem without decreasing excess amounts
    - Increase excess amounts
    - Clear local state of pool
- Take ownership of pool account
    - Rekey account
- Modify validator app
    - Update
    - Delete
- Mix pool accounts in one group
    - Read state from one pool while withdrawing from another
    - Update excess amounts of a different pool
- Exploit incorrect calculations
- Make pool pay unnecessary fees
- Creating Pool Tokens with incorrect information
- Creating Pool Tokens with permissioned functionality enabled (manager, freeze, clawback)

Out of Scope

This is a non exhaustive list of actions that may have detrimental effects for users of the system but are outside the scope of this review.

- Market manipulation
- Front Running
- Freeze or Clawback of pooled assets

Transaction Group Checks

This is a non exhaustive list of checks that have been considered for this review. This includes general checks that apply to all interactions with the contracts and specific checks for each documented operation. The relevant line numbers for the checks in both the LogicSig and Validator App are listed.

General Checks

Check	**LogicSig**	**ValidatorApp**
Pool RekeyTo is not allowed	25:28	
Pool CloseTo is not allowed	15:18	
Pool AssetCloseTo is not allowed	20:23	
Pool LogicSig txns must be grouped with Validator App Call signed by the same LogicSig	30:49	
Pool LogicSig only allows groups with Validator App Calls with arg[0]:		
“bootstrap”	59:63	
“swap”	76:80	
“mint”	90:93	
“burn”	97:100	
“redeem”	103:106	
"fees"	109:112	
Else error	114	
Deny Validator App Update		52:63
Deny Validator App Delete		52:63
Deny Validator App CloseOut		52:63
Allow OptIn to Validator App without		66:73

further side effects		
Allow Validator App creation without further side effects		76:79
Err for any unexpected App Calls		265

Bootstrap

Check	**LogicSig**	**ValidatorApp**
Group Size is 4 (or 5 if asset 2 != Algo)	125:123	
Txn 0		
Type is Payment	686:689 (implicit)	
Receiver is Pool	686:689	
Sender is not Pool	641:644	
Amount >= fee for Txns 1,2,3 (if ASA-Algo)	691:694, 264:270	
Amount >= fee for Txns 1,2,3,4 (if ASA-ASA)	691:694, 302:310	
Txn 1		
Type is AppCall	41:44	
OnCompletion is Optin	52:54	
Sender is Pool	36:39	
App ID is correct	46:49	
Args[1] is asset1	135:138	
Args[2] is asset2	139:142	
Txn 2		
Type is AssetConfig	153:156	
Sender is Pool	147:150	
Asset ID is 0 (new asset)	159:162	
ManagerAddress is ZeroAddress	204:207	
ReserveAddress is ZeroAddress	210:213	

FreezeAddress is ZeroAddress	216:219	
ClawbackAddress is ZeroAddress	222:225	
URL is “https://tinyman.org”	198:201	
Unit Name is “TMPOOL11”	184:187	
Asset Name is “TinymanPool1.1 {asset-1-unit-name}-{asset-2-unit-name }”	191:195	1106:1130
Decimals is 6	172:175	
Total is 0xFFFFFFFFFFFFFFFF	165:169	
Txn 3		
Type is AssetTransfer	235:238	
Sender is Pool	229:232	
Receiver is Pool	247:250	
AssetAmount is 0	253:256	
Asset ID is asset 1	241:244	
Txn 4 (if asset 2 is not Algo)		
Type is AssetTransfer	279:282	
Sender is Pool	275:278	
Receiver is Pool	291:294	
AssetAmount is 0	297:300	
Asset ID is asset 2	285:288	

Mint

Check	**LogicSig**	**ValidatorApp**
Group Size is 5	350:353	
Txn 0		
Type is Payment	686:689 (implicit)	
Receiver is Pool	686:689	
Sender is NotPool	641:644	
Amount >= fee for Txns 1,4	691:694, 421:425	
Txn 1		
Type is AppCall	41:44	
OnCompletion is Noop	67:70	
Sender is Pool	36:39	
App ID is correct	46:49	
Accounts[1] is Pooler	361:364	
Txn 2		
Type is AssetTransfer	372:375 (implicit)	
Sender is not Pool	367:370	
Sender is Pooler	377:380, 416:419	
AssetReceiver is Pool	372:375	
Asset ID is asset 1	383:386	
AssetAmount ?		

Txn 3		
Type is AssetTransfer or Pay	389:397 (implicit)	
Sender is not Pool	377:380, 367:370	
Sender is Pooler	377:380, 416:419	
AssetReceiver or Receiver is Pool	389:397	
Asset ID is asset 2 if AssetTransfer	399:408	
Amount or AssetAmount ?		
Txn 4		
Type is AssetTransfer	416:419 (implicit)	
Sender is Pool	411:414	
AssetReceiver is Pooler	416:419	
Asset ID is liquidity_asset		498:501
AssetAmount <= calculated_liquidity_token_out		476:483

Burn

Check	**LogicSig**	**ValidatorApp**
Group Size is 5	470:473	
Txn 0		
Type is Payment	686:689 (implicit)	
Receiver is Pool	686:689	
Sender is NotPool	641:644	
Amount >= fee for Txns 1,2,3	691:694, 552:558	
Txn 1		
Type is AppCall	41:44	
OnCompletion is Noop	67:70	
Sender is Pool	36:39	
App ID is correct	46:49	
Accounts[1] is Pooler	481:484	
Txn 2		
Type is AssetTransfer	502:505 (implicit)	
Sender is Pool	497:500	
AssetReceiver is Pooler	502:505	
Asset ID is asset 1	508:511	
AssetAmount is < calculated_asset1_out		547:551
Txn 3		

Type is AssetTransfer or Pay	520:528 (implicit)	
Sender is Pool	515:518	
Receiver or AssetReceiver is Pooler	520:528	
Asset ID is asset 2 if AssetTransfer	531:539	
Amount or AssetAmount is < calculated_asset2_out		569:572
Txn 4		
Type is AssetTransfer	547:550 (implicit)	
Sender is not Pool	542:545	
Sender is Pooler	502:505, 520:528	
AssetReceiver is Pool	547:550	
Asset ID is liquidity_asset		498:501
AssetAmount is correct		534:572

Redeem

Check	**LogicSig**	**ValidatorApp**
Group Size is 3	630:633	
Txn 0		
Type is Payment	686:689 (implicit)	
Receiver is Pool	686:689	
Sender is NotPool	680:683	
Amount >= fee for Txns 1,2	652:656	
Txn 1		
Type is AppCall	41:44	
OnCompletion is Noop	67:70	
Sender is Pool	36:39	
App ID is correct	46:49	
Accounts[1] is Pooler/Swapper	642:650	
Txn 2		
Type is Pay or AssetTransfer	642:650 (implicit)	
Sender is Pool	-	
Receiver or AssetReceiver is Pooler/Swapper	642:650	
Amount or AssetAmount <= excess_asset_i_amount		343:345, 93:96
Asset ID is correct		334:342

Amount is correct for Pool+Asset pair		334:342

* “Sender is Pool” is not checked but the sender could not be any other pool because LogicSig 30:49 ensures it will only allow txns with an app call to the validator signed with the same LogicSig.

Swap

Check	**LogicSig**	**ValidatorApp**
Group Size is 4	567:570	
Txn 0		
Type is Payment	686:689 (implicit)	
Receiver is Pool	686:689	
Sender is NotPool	641:644	
Amount >= fee for Txns 1,4	691:694, 618:622	
Txn 1		
Type is AppCall	41:44	
OnCompletion is Noop	67:70	
Sender is Pool	36:39	
App ID is correct	46:49	
Accounts[1] is Swapper	579:582	
Txn 2		
Type is AssetTransfer or Pay	597:605 (implicit)	
Sender is not Pool	585:588	
Sender is Swapper	579:582, 608:616	
Receiver or AssetReceiver is Pool	597:605	
Asset ID is asset 1 or asset 2		837:839, 849:851
Asset ID != Txn 3 Asset ID		837:856
Amount or AssetAmount >= calculated_amount_in if arg[1] == “fo”		929:932, 93:96

Txn 3		
Type is AssetTransfer or Pay	608:616 (implicit)	
Sender is Pool	591:594	
AssetReceiver or Receiver is Swapper	608:616	
Asset ID is asset 1 or asset 2		840:842, 852:854
Asset ID != Txn 2 Asset ID		837:856
Amount or AssetAmount <= calculated_amount_out		1013:1016, 461:464

Redeem Fees

Check	**LogicSig**	**ValidatorApp**
Group Size is 3	664:667	
Txn 0		
Type is Payment	686:689 (implicit)	
Receiver is Pool	686:689	
Sender is NotPool	680:683	
Amount >= fee for Txns 1,2	669:673	
Txn 1		
Type is AppCall	41:44	
OnCompletion is Noop	67:70	
Sender is Pool	36:39	
App ID is correct	46:49	
Txn 2		
Type is AssetTransfer		280:283 (implicit)
AssetReceiver is Creator		280:283
Sender is Pool	*	
Asset ID is liquidity_asset		270:275
AssetAmount <= unclaimed_protocol_fee_token_amount		292:295

* “Sender is Pool” is not checked but the sender could not be any other pool because a) no other pool has the requested liquidity asset and b) LogicSig 30:49 ensures it will only allow txns with an app call to the validator signed with the same LogicSig.

Calculations

This section describes the main calculations that are used in the operations. The variable names here correspond to the names used in the comments of the Validator App source. The line numbers refer to lines of the same source.

Common

```
asset1_supply = asset1_balance - outstanding_asset1_amount [196:199, 157:160, 104:108]

asset2_supply = asset2_balance - outstanding_asset2_amount [203:206, 170:173, 110:138]
liquidity_token = Pool Token (ASA created by Pool representing share of liquidity)
issued_liquidity_tokens = (TOTAL_LIQUIDITY - liquidity_token_balance) +
outstanding_liquidity_token_amount [472:489, 184:187]
```

Mint

```
liquidity_token_amount = gtxn 4 AssetAmount (Pool → Pooler)
MINIMUM_LIQUIDITY = 1000
asset1_amount = gtxn 2 AssetAmount (Pooler → Pool) [93:96]
asset2_amount = gtxn 3 AssetAmount (or Amount) (Pooler → Pool) [461:463]
```

First Mint

```
minted_liquidity_token = liquidity_token_amount + MINIMUM_LIQUIDITY [732:734]
assert(minted_liquidity_token == floor(sqrt(asset1_amount * asset2_amount)))
→ assert(minted_liquidity_token^2 ⩽ asset1_amount^2) [732:752]
and
→ assert((minted_liquidity_token + 1)^2 > asset1_amount * asset2_amount) [756:778]
```

Subsequent Mint

```
A = asset1_amount * issued_liquidity_tokens / asset1_supply [642:652]
B = asset2_amount * issued_liquidity_tokens / asset2_supply [656:666]
calculated_liquidity_token_out = Min(A, B) [671:675]
assert(liquidity_token_amount ⩽ calculated_liquidity_token_out)
→ calculated_liquidity_token_out - liquidity_token_amount (without error) [676:683]
excess_liquidity_token = calculated_liquidity_token_out - liquidity_token_amount [676:684]
```

Burn

```
burn_amount = gtxn 4 AssetAmount
asset1_amount = gtxn 2 AssetAmount (Pooler → Pool) [93:96]
asset2_amount = gtxn 3 AssetAmount (Pooler → Pool) [461:463]

calculated_asset1_out = asset1_supply * (burn_amount / issued_liquidity_tokens) [534:545]
excess_asset_1 = calculated_asset1_out - asset1_amount [547:552]
```

```
assert(asset1_amount ⩽ calculated_asset1_out)
→ calculated_asset1_out - asset1_amount (without error) [547:551]

calculated_asset2_out = asset2_supply * (burn_amount / issued_liquidity_tokens) [555:569]
excess_asset_2 = calculated_asset2_out - asset2_amount [568:573]
assert(asset2_amount ⩽ calculated_asset2_out)
→ calculated_asset2_out - asset2_amount (without error) [568:572]
```

Swap

```
asset_in = gtxn 2 AssetAmount (Swapper → Pool)
asset_out = gtxn 3 AssetAmount (Pool → Swapper)

k = input_supply * output_supply
```

Fixed Input:

```
asset_in_amount_minus_fee = asset_in_amount * 997/1000
calculated_amount_out = output_supply - (k / (input_supply + asset_in_amount_minus_fee))
This can be rewritten to reduce precision loss with integer division:
 calculated_amount_out = (asset_in_amount * 997 * output_supply) / ((input_supply * 1000) +
(asset_in_amount * 997)) [974:999]
assert(asset_out_amount ⩽ calculated_amount_out) [1013:1016]
excess_asset_out = calculated_amount_out - asset_out_amount [1013:1016]
```

Fixed Output:

```
calculated_amount_in = (k / (output_supply - asset_out_amount)) - input_supply
calculated_amount_in_with_fee = calculated_amount_in * 1000/997
This can be rewritten to reduce precision loss with integer division:
calculated_amount_in = ((asset_out_amount * 1000 * input_suppply) / ((output_supply -
asset_out_amount) * 997)) + 1 [891:914]
assert(asset_in_amount ⩾ calculated_amount_in) [929:932]
excess_asset_in = asset_in_amount - calculated_amount_in [929:932]
```

Redeem

```
redeeming_amount = gtxn 2 AssetAmount (Pool → User) [93:96]
key = pool_address + 'e' + assetID [335:340]
excess_asset_i_amount = app_local_get(account=User, key=key) [334:342]
assert(redeeming_amount ⩽ excess_asset_i_amount)
→ excess_asset_i_amount - redeeming_amount (without error) [342:345]
new_excess_asset_i_amount = excess_asset_i_amount - redeeming_amount [342:345]
If new_excess_asset_i_amount > 0: [346:353]
        app_local_put(account=User, key=key, value=new_excess_asset_i_amount) [348]
Else:
        app_local_del(account=User, key=key) [353]
```