SOTERIA

Security Assessment Report

Invariant Protocol v0.1.0

May 6th, 2022

Summary

The Soteria team was engaged to do a thorough security analysis of the Invariant Protocol v0.1.0 Solana smart contract program. The artifact of the audit was the source code of the following on-chain smart contract excluding tests in a private repository:

- Branch `audit`
- Commit `e64141fefef3d3e27e4ed8b9f00585eb47fa744a`

The audit revealed 14 issues including 1 critical vulnerability, which were reported to the Invariant Protocol team.

The Invariant Protocol team responded promptly with a PR for the post-audit review. The scope of the post-audit review is to validate if the reported issues have been addressed. The audit was finalized based on the changes in PR `#189`.

This report describes the findings and resolutions in detail.

Table of Contents

Methodology and Scope of Work

Soteria's audit team, which consists of Computer Science professors and industrial researchers with extensive experience in Solana smart contract security, program analysis, testing and formal verification, performed a comprehensive manual code review, software static analysis and penetration testing.

Assisted by the Soteria Scanner developed in-house, the audit team particularly focused on the following work items:

- Check common security issues.
    - Missing ownership checks
    - Missing signer checks
    - Signed invocation of unverified programs
    - Solana account confusions
    - Arithmetic over- or underflows
    - Numerical precision errors
    - Loss of precision in calculation
    - Insufficient SPL-Token account verification
    - Missing rent exemption assertion
    - Casting truncation
    - Did not follow security best practices
    - Outdated dependencies
    - Redundant code
    - Unsafe Rust code
- Check program logic implementation against available design specifications.
- Check poor coding practices and unsafe behavior.
- The soundness of the economics design and algorithm is out of scope of this work

Result Overview

In total, the audit team found the following issues.

CONTRACT INVARIANT PROTOCOL v0.1.0

Issue	Impact	Status
[C-1] Steal funds/rewards using different lower/upper ticks	Critical	Resolved
[M-1] Incorrect pool key after pool ownership transfer	Medium	Resolved
[M-2] Malicious initial pool tick may prevent users from trading here	Medium	Resolved
[L-1] Arithmetic overflows	Low	Resolved
[L-2] Not resetting the last position after moving	Low	Resolved
[L-3] Inconsistent tick checks	Low	Resolved
[L-4] Missing checks on tick_spacing	Low	Resolved
[L-5] Inconsistent access control in change_protocol_fee	Low	Resolved
[I-1] Inconsistent amount_out and amount_in in swap	Informational	Resolved
[I-2] Redundancies in PDA seeds and account checks	Informational	Resolved
[I-3] Swap exceeds computation limit	Informational	Resolved
[I-4] Oracle can be set but is not used	Informational	Resolved
[I-5] Redundant account and access control	Informational	Resolved
[I-6] Design Choice, best practice and questions	Informational	Resolved

Findings in Detail

IMPACT - CRITICAL

[C-1] Steal funds/rewards using different lower/upper ticks

1. Steal funds in remove_position

Instruction `remove_position` only requires the signature of the `position` owner and accepts the lower and upper ticks from the caller.

However, there is no check to make sure that the lower and upper ticks are the same ones that were used to create the position. As a result, malicious users provide a wider tick range and steal funds.

```
/* programs/invariant/src/instructions/remove_position.rs */
015: #[derive(Accounts)]
016: #[instruction(index: i32, lower_tick_index: i32, upper_tick_index: i32)]
017: pub struct RemovePosition<'info> {
050:     #[account(mut,
051:         seeds = [b"tickv1", pool.key().as_ref(), &lower_tick_index.to_le_bytes()],
052:         bump = lower_tick.load()?.bump
053:     )]
054:     pub lower_tick: AccountLoader<'info, Tick>,
055:     #[account(mut,
056:         seeds = [b"tickv1", pool.key().as_ref(), &upper_tick_index.to_le_bytes()],
057:         bump = upper_tick.load()?.bump
058:     )]
059:     pub upper_tick: AccountLoader<'info, Tick>,

118: impl<'info> RemovePosition<'info> {
119:     pub fn handler(
124:     ) -> ProgramResult {
138:         let (amount_x, amount_y) = {
145:             let (amount_x, amount_y) = removed_position.modify(
147:                 upper_tick,
148:                 lower_tick,
152:             )?;
160:             (amount_x, amount_y)
161:         };
218:         token::transfer(self.send_x().with_signer(signer), amount_x.0)?;
219:         token::transfer(self.send_y().with_signer(signer), amount_y.0)?;
```

PoC. As shown in the log below, an attacker created a position using ticks -2/2. And, `amount_x` and `amount_y` are both 4000. When removing this position, the attacker uses a wider tick range (-20/20 as the lower/higher ticks) and gets back 39978 `amount_x` and 39978 `amount_y`.

```
[2022-04-08T03:59:45.585955000Z DEBUG solana_runtime::message_processor::stable_log] Program FFtYJqUdvZwJkjx4YCTV61ik8rUY3HAp4dMzNkvV76Nx invoke [1]
[2022-04-08T03:59:45.588229000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: CreatePosition
[2022-04-08T03:59:45.597309000Z DEBUG solana_runtime::message_processor::stable_log] Program 11111111111111111111111111111111 invoke [2]
[2022-04-08T03:59:45.597422000Z DEBUG solana_runtime::message_processor::stable_log] Program 11111111111111111111111111111111 success
[2022-04-08T03:59:45.601200000Z DEBUG solana_runtime::message_processor::stable_log] Program log: INVARIANT: CREATE POSITION
[2022-04-08T03:59:45.601981000Z DEBUG solana_runtime::message_processor::stable_log] Program log: position_list.head: 1
[2022-04-08T03:59:45.602495000Z DEBUG solana_runtime::message_processor::stable_log] Program log: position_list.head new: 2
[2022-04-08T03:59:45.605337000Z DEBUG solana_runtime::message_processor::stable_log] Program log: max_liquidity_per_tick: 170141183460469231731687303 7.158841
[2022-04-08T03:59:45.605731000Z DEBUG solana_runtime::message_processor::stable_log] Program log: lower_tick.update: -2
[2022-04-08T03:59:45.606310000Z DEBUG solana_runtime::message_processor::stable_log] Program log: self.liquidity_gross: 30000000000.0
[2022-04-08T03:59:45.606923000Z DEBUG solana_runtime::message_processor::stable_log] Program log: liquidity_delta: 40000000.0
[2022-04-08T03:59:45.641322000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
[2022-04-08T03:59:45.642169000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: Transfer
[2022-04-08T03:59:45.643904000Z DEBUG solana_rbpf::vm] BPF instructions executed (interp): 3395
[2022-04-08T03:59:45.643943000Z DEBUG solana_rbpf::vm] Max frame depth reached: 6
[2022-04-08T03:59:45.643961000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 3495 of 1307094 compute units
[2022-04-08T03:59:45.644182000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
[2022-04-08T03:59:45.659065000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
[2022-04-08T03:59:45.659878000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: Transfer
[2022-04-08T03:59:45.661626000Z DEBUG solana_rbpf::vm] BPF instructions executed (interp): 3395
[2022-04-08T03:59:45.661661000Z DEBUG solana_rbpf::vm] Max frame depth reached: 6
[2022-04-08T03:59:45.661678000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 3495 of 1300218 compute units
[2022-04-08T03:59:45.661865000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
[2022-04-08T03:59:45.662526000Z DEBUG solana_runtime::message_processor::stable_log] Program log: amount_x.0: 4000
[2022-04-08T03:59:45.662782000Z DEBUG solana_runtime::message_processor::stable_log] Program log: amount_y.0: 4000

[2022-04-08T03:59:45.721110000Z DEBUG solana_runtime::message_processor::stable_log] Program FFtYJqUdvZwJkjx4YCTV61ik8rUY3HAp4dMzNkvV76Nx invoke [1]
[2022-04-08T03:59:45.723153000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: RemovePosition
[2022-04-08T03:59:45.732119000Z DEBUG solana_runtime::message_processor::stable_log] Program log: INVARIANT: REMOVE POSITION
[2022-04-08T03:59:45.732664000Z DEBUG solana_runtime::message_processor::stable_log] Program log: position_list.head: 2
[2022-04-08T03:59:45.733046000Z DEBUG solana_runtime::message_processor::stable_log] Program log: lower_tick.index: -20
[2022-04-08T03:59:45.733289000Z DEBUG solana_runtime::message_processor::stable_log] Program log: upper_tick.index: 20
[2022-04-08T03:59:45.733918000Z DEBUG solana_runtime::message_processor::stable_log] Program log: liquidity_delta: 40000000.0
[2022-04-08T03:59:45.736689000Z DEBUG solana_runtime::message_processor::stable_log] Program log: max_liquidity_per_tick: 170141183460469231731687303 7.158841
[2022-04-08T03:59:45.737205000Z DEBUG solana_runtime::message_processor::stable_log] Program log: lower_tick.update: -20
[2022-04-08T03:59:45.737776000Z DEBUG solana_runtime::message_processor::stable_log] Program log: self.liquidity_gross: 20000000000.0
[2022-04-08T03:59:45.738361000Z DEBUG solana_runtime::message_processor::stable_log] Program log: liquidity_delta: 40000000.0
[2022-04-08T03:59:45.777419000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
[2022-04-08T03:59:45.778269000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: Transfer
[2022-04-08T03:59:45.779650000Z DEBUG solana_rbpf::vm] BPF instructions executed (interp): 3395
[2022-04-08T03:59:45.779701000Z DEBUG solana_rbpf::vm] Max frame depth reached: 6
[2022-04-08T03:59:45.779723000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 3495 of 1305379 compute units
[2022-04-08T03:59:45.779919000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
[2022-04-08T03:59:45.794108000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
[2022-04-08T03:59:45.794821000Z DEBUG solana_runtime::message_processor::stable_log] Program log: Instruction: Transfer
[2022-04-08T03:59:45.796088000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 3495 of 1298612 compute units
[2022-04-08T03:59:45.796260000Z DEBUG solana_runtime::message_processor::stable_log] Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
[2022-04-08T03:59:45.796805000Z DEBUG solana_runtime::message_processor::stable_log] Program log: amount_x.0: 39978
[2022-04-08T03:59:45.797032000Z DEBUG solana_runtime::message_processor::stable_log] Program log: amount_y.0: 39978
```

2. Steal funds in claim_fee

Similar to the previous issue in `remove_position`, the position owner can provide arbitrary lower/upper ticks and sign the instruction `claim_fee`.

Since there are no checks on the provided ticks, malicious users may be able to manipulate `position.tokens_owed_x` and `position.tokens_owed_y` to steal money.

```
/* programs/invariant/src/instructions/claim_fee.rs */
013: #[derive(Accounts)]
014: #[instruction( index: u32, lower_tick_index: i32, upper_tick_index: i32)]
015: pub struct ClaimFee<'info> {
030:     #[account(mut,
031:         seeds = [b"tickv1", pool.key().as_ref(), &lower_tick_index.to_le_bytes()],
032:         bump = lower_tick.load()?.bump
033:     )]
034:     pub lower_tick: AccountLoader<'info, Tick>,
035:     #[account(mut,
036:         seeds = [b"tickv1", pool.key().as_ref(), &upper_tick_index.to_le_bytes()],
037:         bump = upper_tick.load()?.bump
038:     )]
039:     pub upper_tick: AccountLoader<'info, Tick>,

097: impl<'info> ClaimFee<'info> {
098:     pub fn handler(&self) -> ProgramResult {
110:         position
111:             .modify(
113:                 upper_tick,
114:                 lower_tick,
118:             )
121:         let fee_to_collect_x = TokenAmount::from_decimal(position.tokens_owed_x);
122:         let fee_to_collect_y = TokenAmount::from_decimal(position.tokens_owed_y);
133:         token::transfer(cpi_ctx_x, fee_to_collect_x.0)?;
134:         token::transfer(cpi_ctx_y, fee_to_collect_y.0)?;
```

3. Steal reward via update_seconds_per_liquidity

Similarly, in instruction `update_seconds_per_liquidity`, malicious users can provide lower and upper ticks that are different from the ones when the position was created. They can manipulate `position.seconds_per_liquidity_inside` with a wider tick range as well as the reward that is calculated based on the `position.seconds_per_liquidity_inside`.

```
/* programs/invariant/src/instructions/update_seconds_per_liquidity.rs */
12: #[derive(Accounts)]
13: #[instruction(lower_tick_index: i32, upper_tick_index: i32, index: i32)]
14: pub struct UpdateSecondsPerLiquidity<'info> {
20:     #[account(
21:         seeds = [b"tickv1", pool.key().as_ref(), &lower_tick_index.to_le_bytes()],
22:         bump = lower_tick.load()?.bump
23:     )]
24:     pub lower_tick: AccountLoader<'info, Tick>,
25:     #[account(
26:         seeds = [b"tickv1", pool.key().as_ref(), &upper_tick_index.to_le_bytes()],
27:         bump = upper_tick.load()?.bump
28:     )]
29:     pub upper_tick: AccountLoader<'info, Tick>,
45: }

48:     pub fn handler(&self) -> ProgramResult {
56:         position.seconds_per_liquidity_inside =
57:             calculate_seconds_per_liquidity_inside(lower_tick, upper_tick, ...);
```

Resolution

Because the contract has been deployed, we immediately reported our findings with PoCs to the Invariant Protocol team. The team promptly confirmed and fixed the issues.

We did not review the historical transactions, as it's not in the scope of this audit. However, the Invariant Protocol team confirmed that no one had exploited this vulnerability.

IMPACT – MEDIUM

[M-1] Incorrect pool key after pool ownership transfer

There is a copy & paste error at line 67 in transfer_position_ownership.rs. After transfer, the pool key will be lost. It should be pool: removed_position.pool.

```
/* programs/invariant/src/instructions/transfer_position_ownership.rs */
65:             **new_position = Position {
66:                 owner: *self.recipient.key,
67:                 pool: *self.recipient.key,
68:                 id: removed_position.id,
79:             };
```

Resolution

The Invariant team confirmed and fixed this issue.

IMPACT - MEDIUM

[M-2] Malicious initial pool tick may prevent users from trading here

Anyone can call the instruction `create_pool` to create a pool. However, since the `seeds` for the pool PDA contains the token pair `token_x` and `token_y`, once a pool for a particular token pair and `fee_tier` is created, it's not possible to create another pool for the same token pair and `fee_tier`. In addition, when creating the pool, the caller needs to provide the initial tick index, which will determine the initial price.

Malicious users may create pools for many token pairs and provide large `init_tick`. As a result, the initial price will be extremely unfair. Since others cannot create new pools for the same token pairs, users may not want to trade due to the unfair price.

```
/* programs/invariant/src/instructions/create_pool.rs */
016: #[derive(Accounts)]
017: pub struct CreatePool<'info> {
018:     #[account(seeds = [b"statev1".as_ref()], bump = state.load()?.bump)]
019:     pub state: AccountLoader<'info, State>,
020:     #[account(init,
021:         seeds = [b"poolv1", token_x.to_account_info().key.as_ref(),
                              token_y.to_account_info().key.as_ref(),
                              &fee_tier.load()?.fee.v.to_le_bytes(),
                              &fee_tier.load()?.tick_spacing.to_le_bytes()],
022:         bump, payer = payer
023:     )]
024:     pub pool: AccountLoader<'info, Pool>,
046:     #[account(mut)]
047:     pub payer: Signer<'info>,
054: }

056: impl<'info> CreatePool<'info> {
057:     pub fn handler(&self, init_tick: i32, bump: u8) -> ProgramResult {
076:         **pool = Pool {
085:             sqrt_price: calculate_price_sqrt(init_tick),
086:             current_tick_index: init_tick,
100:         };
```

Resolution

The Invariant team is aware of such behaviors and does not consider this scenario as an issue. This is an intended behavior.

Non-market prices always make it possible to swap with profit and those swaps change the price towards the market one. The case where there is no liquidity can be easily solved by providing a minimum amount of liquidity in full rage. Then, the price will move to the market one in a few transactions, which is considered to be the correct permission-less trade-off.

IMPACT – LOW

[L-1] Arithmetic overflows

1. The type of `owner_list.head` and `recipient_list.head` is `u32`.

```
/* programs/invariant/src/instructions/transfer_position_ownership.rs */
60:         owner_list.head -= 1;
61:         recipient_list.head += 1;
```

2. The type of `position_iterator` is `u128`.

```
/* programs/invariant/src/structs/position.rs */
112:     pub fn initialized_id(&mut self, pool: &mut Pool) {
113:         self.id = pool.position_iterator;
114:         pool.position_iterator += 1;
115:     }
```

3. `tick` and `tick_spacing` are integers.

```
/* programs/invariant/src/structs/tickmap.rs */
39: pub fn get_search_limit(tick: i32, tick_spacing: u16, up: bool) -> i32 {
40:     let index = tick / tick_spacing as i32;
/* programs/invariant/src/structs/tickmap.rs */
85: let (mut byte, mut bit) = tick_to_position(tick + tick_spacing as i32, tick_spacing);
```

4. `fee_protocol_token_y` and `fee_protocol_token_x` are `u64`.

```
/* programs/invariant/src/structs/pool.rs */
36:     pub fn add_fee(&mut self, amount: TokenAmount, in_x: bool) {
45:         if in_x {
48:             self.fee_protocol_token_x += protocol_fee.0;
49:         } else {
52:             self.fee_protocol_token_y += protocol_fee.0;
53:         }
54: }
```

5. `current_timestamp` and `self.last_timestamp` are `u64`.

```
70:     pub fn update_seconds_per_liquidity_global(&mut self, current_timestamp: u64) {
71:         self.seconds_per_liquidity_global = self.seconds_per_liquidity_global
72:             + (FixedPoint::from_integer((current_timestamp - self.last_timestamp) as u128)
73:                 / self.liquidity);
```

6. `tick_index` is `i32`.

```
/* programs/invariant/src/instructions/swap.rs  */
221: pool.current_tick_index = if x_to_y && is_enough_amount_to_cross {
222:     tick_index - pool.tick_spacing as i32
```

As a fix, it may be a good idea to enable the overflow runtime check in `Cargo.toml`

```
[profile.release]
overflow-checks = true
```

Resolution

The Invariant team confirmed and fixed the issues.

IMPACT – LOW

[L-2] Not resetting the last position after moving

In instruction `transfer_position_ownership`, after moving the last position to the one to be deleted, the last position is not reset properly.

```
/* programs/invariant/src/instructions/transfer_position_ownership.rs */
082:         // when removed position is not the last one
083:         if owner_list.head != index {
084:             let last_position = self.last_position.load_mut()?;
085:
086:             **removed_position = Position {
087:                 owner: last_position.owner,
100:             };
101:         }
```

In particular, after line 100, `last_position` should be reset like what `remove_position` does

```
/* programs/invariant/src/instructions/remove_position.rs */
192:
193:         // when removed position is not the last one
194:         if position_list.head != index {
195:             let mut last_position = self.last_position.load_mut()?;
198:             **removed_position = Position {
199:                 bump: removed_position.bump,
200:                 owner: last_position.owner,
212:             };
214:             *last_position = Default::default();
215:         }
```

Resolution

The Invariant team confirmed and fixed the issue.

IMPACT – LOW

[L-3] Inconsistent tick checks

It's unclear if `tick` can be `MAX_TICK`.

```
/* programs/invariant/src/math.rs */
22:     assert!(tick <= MAX_TICK, "tick over bounds");

/* programs/invariant/src/util.rs */
35:     require!(tick_index > (-MAX_TICK), InvalidTickIndex);
36:     require!(tick_index < MAX_TICK, InvalidTickIndex);
```

Resolution

The Invariant team confirmed and fixed the issue.

IMPACT – LOW

[L-4] Missing checks on tick_spacing

Although instruction `create_fee_tier` is privileged, `tick_spacing` should still be checked because it cannot be `0`. However, the non-zero check is currently missing.

```
/* programs/invariant/src/instructions/create_fee_tier.rs */
25: impl<'info> CreateFeeTier<'info> {
26:     pub fn handler(&self, fee: u128, tick_spacing: u16, bump: u8) -> ProgramResult {
32:         **fee_tier = FeeTier {
33:             fee,
34:             tick_spacing,
35:             bump,
36:         };
39:     }
40: }
```

Resolution

The Invariant team confirmed and fixed the issue.

IMPACT – LOW

[L-5] Inconsistent access control in change_protocol_fee

```
/* programs/invariant/src/lib.rs */
124:     #[access_control(receiver(&ctx.accounts.pool, &ctx.accounts.admin))]
125:     pub fn change_protocol_fee(
128:     ) -> ProgramResult {
129:         ctx.accounts.handler(protocol_fee)
130:     }
```

The `#[access_control` makes sure that `pool.fee_receiver` is the `admin`. This works before changing the `pool.fee_receiver` via instruction `change_fee_receiver`.

If the `fee_receiver` is set to a non-admin account, the access control cannot be satisfied so this instruction will always fail.

Resolution

The Invariant team confirmed that this is an intended behavior.

IMPACT – INFO

[I-1] Inconsistent amount_out and amount_in in swap

In src/math.rs, amount_out is set to amount (smaller) but amount_in is not changed, which seems to lead to inconsistency.

```
/* programs/invariant/src/math.rs */
158:     // Amount out can not exceed amount
159:     if !by_amount_in && amount_out > amount {
160:         amount_out = amount;
161:     }
```

Resolution

The Invariant team was aware of this issue and it's the intended behavior right now.

IMPACT – INFO

[I-2] Redundancies in PDA seeds and account checks

1. The constraint in line 46 implies the ones in lines 44-45 will be true because they were checked when creating the pool. It's not a big issue and still safe.
   Instruction withdraw_protocol_fee has similar issues.

```
/* programs/invariant/src/instructions/swap.rs */
15: #[derive(Accounts)]
16: pub struct Swap<'info> {
43:     #[account(mut,
44:         constraint = &reserve_x.mint == token_x.to_account_info().key @ InvalidMint,
45:         constraint = &reserve_x.owner == program_authority.key @ InvalidAuthority,
46:         constraint = reserve_x.to_account_info().key == &pool.load()?.token_x_reserve
                        @ InvalidTokenAccount
47:     )]
48:     pub reserve_x: Box<Account<'info, TokenAccount>>,
```

2. create_program_address will hash both the seeds and program_id. It's not necessary to put another program_id in the seeds. It's not incorrect as it's still safe.

```
/* programs/invariant/src/instructions/create_fee_tier.rs */
8: #[derive(Accounts)]
9: #[instruction(fee: u128, tick_spacing: u16)]
10: pub struct CreateFeeTier<'info> {
11:     #[account(init,
12:         seeds = [b"feetierv1", program_id.as_ref(), &fee.to_le_bytes(),
                    &tick_spacing.to_le_bytes()],
14:     )]
```

Resolution

Since it's still correct and safe, no actions will be taken.

IMPACT – INFO

[I-3] Swap exceeds computation limit

At line 137 of swap.rs, the loop iterates on the ticks until remaining_amount is zero. Depending on how the ticks are distributed, it can easily exceed the computing limit.

```
/* programs/invariant/src/instructions/swap.rs */
137:        while !remaining_amount.is_zero() {
237:        }
```

For example, we got both exceeded maximum number of instructions allowed (1400000) and out of memory errors in our tests. In addition, we found the loop keeps going even the liquidity becomes 0, which may be used to further optimize the process.

```
Program log: INVARIANT: SWAP
...
Program log: tick_index: -2
Program log: tick_address: DhutfvqscYjfN7KGcG9vs8GVrhBzmVuaBihm96JPwujj
Program log: cross_tick pool.liquidity: Liquidity { v: 19960000000000000 }

Program log: pool.current_tick_index: -3
Program log: remaining_amount: TokenAmount(19945005500)
Program log: pool.liquidity: Liquidity { v: 19960000000000000 }
...
Program log: pool.current_tick_index: -21
Program log: remaining_amount: TokenAmount(19927052275)
Program log: pool.liquidity: Liquidity { v: 0 }
...
Program log: pool.current_tick_index: -278
Program log: remaining_amount: TokenAmount(19927052275)
Program log: pool.liquidity: Liquidity { v: 0 }
...
Program log: pool.current_tick_index: -535
Program log: remaining_amount: TokenAmount(19927052275)
Program log: pool.liquidity: Liquidity { v: 0 }
...
Program log: pool.current_tick_index: -6446
Program log: remaining_amount: TokenAmount(19927052275)
Program log: pool.liquidity: Liquidity { v: 0 }
Program log: Error: memory allocation failed, out of memory
Program FFtYJgUdvZwJkjx4YCTV61ik8rUY3HAp4dMzNkvV76Nx consumed 935856 of 1400000 compute units
Program failed to complete: BPF program panicked
Program FFtYJgUdvZwJkjx4YCTV61ik8rUY3HAp4dMzNkvV76Nx failed: Program failed to complete
```

Resolution

Improvements to the logarithm computation and the zero-liquidity scenario have been implemented.

IMPACT – INFO

[I-4] Oracle can be set but is not used

Oracle can be set using the instruction `initialize_oracle`. However, it's not used anywhere. Is this an incomplete feature?

```
/* programs/invariant/src/instructions/initialize_oracle.rs */
28: impl<'info> InitializeOracle<'info> {
29:     pub fn handler(&self) -> ProgramResult {
30:         msg!("INVARIANT: INITIALIZE ORACLE");
40:         pool.set_oracle(self.oracle.key());
41:         oracle.init();
44:     }
45: }

/* programs/invariant/src/structs/pool.rs */
34: impl Pool {
78:     pub fn set_oracle(&mut self, address: Pubkey) {
79:         self.oracle_address = address;
81:     }
82: }
```

Resolution

The Invariant Protocol team confirmed that this is for future integration.

IMPACT – INFO

[I-5] Redundant account and access control

1. `program_authority` is not used in this instruction and seems redundant

```
/* programs/invariant/src/instructions/change_fee_receiver.rs */
06: #[derive(Accounts)]
07: pub struct ChangeFeeReceiver<'info> {
19:     #[account(constraint = &state.load()?.admin == admin.key @ InvalidAdmin)]
20:     pub admin: Signer<'info>,
22:     #[account(constraint = &state.load()?.authority == program_authority.key @ InvalidAuthority)]
23:     pub program_authority: AccountInfo<'info>,
24: }
25:

/* programs/invariant/src/lib.rs */
132:     #[access_control(admin(&ctx.accounts.state, &ctx.accounts.admin))]
133:     pub fn change_fee_receiver(ctx: Context<ChangeFeeReceiver>) -> ProgramResult {
134:         ctx.accounts.handler()
135:     }
```

2. The following is what the `#[access_control` does at line 132. It checks the same condition as the constraints in line 19.

```
// #[access_control(admin(&ctx.accounts.state, &ctx.accounts.admin))]
fn admin(state_loader: &AccountLoader<State>, signer: &AccountInfo) -> Result<()> {
    let state = state_loader.load()?;
    if !(signer.key.eq(&state.admin)) {
        return Err(crate::ErrorCode::Unauthorized.into());
    };
    Ok(())
}
```

Resolution

The Invariant Protocol team confirmed and removed `program_authority`.

The extra access control check is acceptable.

IMPACT – INFO

[I-6] Design Choice, best practice, and questions

1. In instruction `remove_position`, a tick is closed when liquidity is zero. Why does the program need to close ticks, given they can be created by anyone without providing liquidity?

2. In instruction `remove_position`, the `owner` who receives the remaining SOL in tick accounts may not be the one who paid to create them. Is this an intended behavior?

3. In `create_pool`, the `protocol_fee` is set to `0.2`, which seems too high.

```
/* programs/invariant/src/instructions/create_pool.rs */
83:             protocol_fee: FixedPoint::from_scale(2, 1),
```

Resolution

The Invariant Protocol team confirmed these are intended or allowed behaviors.

DISCLAIMER

The instance report ("Report") was prepared pursuant to an agreement between Coderrect Inc. d/b/a Soteria ("Company") and Akudama GmbH ("Client"). This Report solely includes the results of a technical assessment of a specific build and/or version of the Client's code specified in the Report ("Assessed Code") by the Company. The sole purpose of the Report is to provide the Client with the results of the technical assessment of the Assessed Code. The Report does not apply to any other version and/or build of the Assessed Code. Regardless of the contents of the Report, the Report does not (and should not be interpreted to) provide any warranty, representation or covenant that the Assessed Code: (i) is error and/or bug free, (ii) has no security vulnerabilities, and/or (iii) does not infringe any third-party rights. Moreover, the Report is not, and should not be considered, an endorsement by the Company of the Assessed Code and/or of the Client. Finally, the Report should not be considered investment advice or a recommendation to invest in the Assessed Code and/or the Client.

This Report is considered null and void if the Report (or any portion thereof) is altered in any manner.

ABOUT

Founded by leading academics in the field of software security and senior industrial veterans, Soteria is a leading blockchain security company that currently focuses on Solana programs. We are also building sophisticated security tools that incorporate static analysis, penetration testing, and formal verification.

At Soteria, we identify and eliminate security vulnerabilities through the most rigorous process and aided by the most advanced analysis tools.

For more information, check out our website and follow us on twitter.