GooseFX SSL

Audit

Presented by:

OtterSec contact@osec.io

Robert Chen notdeghost@osec.io

William Wang defund@osec.io

Contents

Appendices

01 | Executive Summary

Overview

GooseFX engaged OtterSec to perform an assessment of the GFX SSL and GFX Controller programs. This assessment was conducted between August 8th and August 26th, 2022.

Critical vulnerabilities were communicated to the team prior to the delivery of the report to speed up remediation. After delivering our audit report, we worked closely with the team over to streamline patches and confirm remediation.

We delivered final confirmation of the patches August 31st, 2022.

Key Findings

The following is a summary of the major findings in this audit.

- 6 findings total
- 2 vulnerabilities which could lead to loss of funds
  - OS-GFX-ADV-00: SSL deposits and withdrawals rely on an admin to consistently update swapped liability metrics, but there is no on-chain logic which asserts this has been done.
  - OS-GFX-ADV-01: Controller unstake rounds in the incorrect direction when computing how many shares should be removed.

02 | Scope

The source code was delivered to us in a git repository at github.com/GooseFX1/gfx-ssl. This audit was performed against commit `cf0d47a`.

There were a total of 2 programs included in this audit. A brief description of the programs is as follows. A full list of program files and hashes can be found in Appendix A.

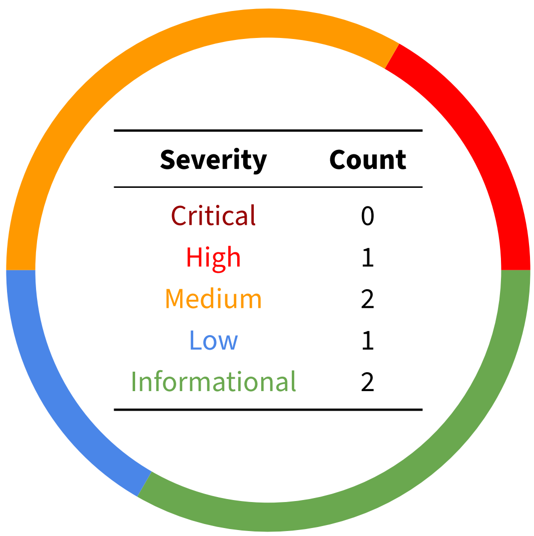
Name	Description
gfx-controller	Manages controller accounts, which serve as the master of other GFX programs, and governance token staking.
gfx-ssl	Liquidity vault which performs market-making through via single-sided liquidity (SSL) pools. During a swap, pools temporarily form a pair to facilitate transfer.

03 | Findings

Overall, we report 6 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings don't have an immediate impact but will help mitigate future vulnerabilities.

The below chart displays the findings by severity.



Severity	Count
Critical	0
High	1
Medium	2
Low	1
Informational	2

04 | Vulnerabilities

Here we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have **immediate** security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix D.

ID	Severity	Status	Description
OS-GFX-ADV-00	High	Resolved	SSL deposits and withdrawals rely on an admin to consistently update swapped liability metrics, but there is no on-chain logic which asserts this has been done.
OS-GFX-ADV-01	Medium	Resolved	Controller unstake rounds in the incorrect direction when computing how many shares should be removed.
OS-GFX-ADV-02	Medium	Resolved	SSL withdraw rounds in the incorrect direction when computing how much liability is locked as pool tokens.
OS-GFX-ADV-03	Low	Resolved	Unchecked arithmetic allows for integer overflows, which can create invalid state.

OS-GFX-ADV-00 [high] [resolved] | Unsound crank mechanism

The gfx-ssl program relies on the admin consistently invoking the `CrankLiability` instruction to update the `swapped_liability_native` field of each SSL pool. This quantity represents the value, i.e. the equivalent number of risk tokens, of all other swapped assets.

The issue is that these quantities do not necessarily reflect the current state of the pool. Due to natural price fluctuations, failure to update swapped liabilities can lead to arbitrage opportunities in deposit/withdrawal. Furthermore, since the `Pair::curve_swap` method neglects to adjust swapped liabilities, the following attack is possible:

1. Perform a massive swap to increase the swapped liability in an SSL pool.
2. Before the admin invokes the `CrankLiability` instruction, deposit liquidity into the SSL pool. Shares will be awarded based on an outdated view of the swapped liability.
3. After the admin invokes the `CrankLiability` instruction, withdraw liquidity from the SSL pool. Risk tokens will be transferred based on the correct view of the swapped liability; this is an unfairly increased percentage of the pool.

Remediation

In the `SSL` struct, add a new field which records when `swapped_liability_native` was last updated. In the `CrankLiability` instruction, overwrite the fields of each updated SSL pool with the current timestamp. In instructions which reference total or swapped liability, verify that SSL pools have been recently updated.

In the `Snapshot::apply` method, use the oracle price and transferred token amounts to calculate `swapped_liability_native` for both pools. Since transfers in `Pair::rebalance_swap` and `Pair::curve_swap` are both taken into consideration, the existing logic shown below will be invalidated.

gfx-ssl/src/states/pair/swap.rs RUST

```rust
// also reduce the swapped liability native because rebalance is returning the swapped liability back to the pool
let delta_swapped_liability = snapshot
    .swapped_liabilities_native
    .0
    .min(U64(amount_in_wo_fee));
snapshot.swapped_liabilities_native.0 -= delta_swapped_liability;
```

gfx-ssl/src/states/pair/swap.rs RUST

```rust
pub fn apply(&self, ssl0: &mut SSL, ssl1: &mut SSL, pair: &mut Pair) {
    ssl0.swapped_liability_native = *self.swapped_liabilities_native.0;
    ssl1.swapped_liability_native = *self.swapped_liabilities_native.1;
```

Patch

Fixed in 69c0513, 303bd06, and 1f24518.

OS-GFX-ADV-01 [med] [resolved] | Incorrect rounding in unstake

Description

In the gfx-controller program's `Unstake` instruction, a user withdraws a percentage of their share from the pool. There are three quantities the program computes with this percentage:

- `delta_share`, the number of shares to remove;
- `delta_balance`, the number of tokens, including staking rewards, to transfer;
- `delta_staked_amount`, the number of tokens, excluding staking rewards, to transfer.

gfx-controller/src/contexts/unstake.rs RUST

```rust
let delta_share =
    U128::from(staking_account.share) * unstake_percent / 10u64.pow(BP_DECIMAL);
let delta_balance =
    U128::from(controller.staking_balance) * staking_account.share * unstake_percent
        / controller.total_staking_share
        / 10u64.pow(BP_DECIMAL);
let delta_staked_amount =
    U128::from(staking_account.amount_staked) * unstake_percent / 10u64.pow(BP_DECIMAL);
```

The issue is that all three computations use a floor division. In particular, consider a scenario where `unstake_percent` is tuned such that `delta_share` is floored zero and `delta_balance` is non-zero. In this case, the user receives tokens without sacrificing any shares.

Remediation

Calculate `delta_balance` based on `delta_share`, instead of `unstake_percent` directly. With this change, a floor division will always work in favor of the pool.

RUST

```rust
let delta_balance = U128::from(controller.staking_balance) * delta_share / controller.total_staking_share;
```

Patch

Fixed in c858099.

OS-GFX-ADV-02 [med] [resolved] | Incorrect rounding in withdraw

Description

In the gfx-ssl program's `Withdraw` instruction, a user withdraws a percentage of their share from the pool. However, the user may also have a portion of their shares locked as pool tokens. In this case, their withdrawal is capped to their unlocked liability. This is enforced by checking that `user_total_balance` is greater than the sum of two quantities:

- `delta_rt`, the number of risk tokens to transfer;
- `liquidity_account.pt_minted`, the number of minted pool tokens. Since risk and pool tokens can use different decimals, this quantity is converted into the risk token decimal.

gfx-ssl/src/state/ssl/withdraw.rs RUST

```rust
let user_total_balance = total_liability * liquidity_account.share / ssl.total_share;
require_gte!(
    user_total_balance,
    delta_rt
        + scale_decimal(
            liquidity_account.pt_minted,
            POOL_TOKEN_DECIMALS as u8,
            ssl.decimals
        )
); // make sure pt_minted <= user_total_balance after withdraw
```

Notice that if the pool token decimal is greater than that of the risk token, `scale_decimal` acts as a floor division. However, this is the incorrect rounding direction; rounding down allows the user to underrepresent how much value is locked as pool tokens. Consequently, they can withdraw more than allowed.

Remediation

Replace the call to `scale_decimal` with a new variant which rounds up.

Patch

Fixed in a344a8e.

OS-GFX-ADV-03 [low] [resolved] | Unchecked arithmetic

Description

The gfx-ssl program often uses unchecked arithmetic, which has the possibility of overflowing. One such occurrence is in the `Pair::rebalance_swap` method, when the program computes how many tokens the user should transfer. Before line 223, `amount_in` is guaranteed to be greater than `fee`. However, it is not guaranteed to be greater than `max_in` (due to the quantity being rounded up in `ssl_in.descale_up`). Thus when the minimum between both quantities are taken on line 223, it is possible for `amount_in` to be greater than `fee`. In this case, `amount_in_wo_fee` will be an unreasonably large value.

gfx-ssl/state/pair/swap.rs RUST

```rust
216 // Recalculate amount_in and fee. The recalculated amount_in might be less than max_in
217 // due to deplete of surplus
218 let amount_in_wo_fee = ssl_in.descale_up(ssl_out.to_decimal(amount_out) / oracle_price);
219 let (amount_in, fee) = amount_in_wo_fee.combine_fee(fee_rate);
220 let (fee_to_lp, fee_to_platform) = fee.split_fee(platform_fee_rate);
221 // it is possible that amount_in is larger than max_in due to rounding error
222 // so we fix it here
223 let amount_in = amount_in.min(max_in);
224 let amount_in_wo_fee = amount_in - fee;
```

Other instances of unchecked arithmetic are protected by implicit bounds on the operands. However, in general it is safe practice to always use checked arithmetic.

Remediation

There are two approaches to avoid unchecked arithmetic.

- Explicitly use checked arithmetic operations, such as `checked_add` and `checked_sub`.
- The program already makes use of an U64 type, which performs checked arithmetic on a wrapped u64 value. Casting u64 quantities to U64 would eliminate the potential for arithmetic overflow.

Additionally, consider appending the following in `Cargo.toml`:

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

This will enable integer overflow checks by default in release builds.

Patch

Fixed in 19f7a88 and 83e54eb.

05 | General Findings

Here we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they do represent antipatterns and could introduce a vulnerability in the future.

ID	Status	Description
OS-GFX-SUG-00	Resolved	Charging fees during controller unstake is unnecessary.
OS-GFX-SUG-01	Resolved	Curve arithmetic should be refactored to be simpler.

OS-GFX-SUG-00 [resolved] | Unstake fees are unnecessary

Description

In the gfx-controller program's `Unstake` instruction, fees are charged as a proportion of the staking reward bonus.

gfx-controller/src/contexts/unstake.rs RUST

```rust
let to_fee_collector = U128(delta_balance.saturating_sub(*delta_staked_amount))
    * controller.withdraw_fee as u64
    / BP_SCALE;
```

This is unnecessary, as staking rewards are already distributed by the admin in a controlled fashion. In other words, one could effectively charge the same fee by reducing how much is dealt in the first place.

gfx-controller/src/states/controller.rs RUST

```rust
let now = Clock::get()?.unix_timestamp;
let elapsed = now.checked_sub(self.last_distribution_time).unwrap();
let reward_to_distribute = U128::from(elapsed as u64) * self.daily_reward / SECONDS_PER_DAY;
```

Remediation

Remove the fee collection functionality from the `Unstake` instruction. If the admin would like to collect earnings from the controller vault, modify the `DistributeStakingReward` instruction to additionally transfer tokens to an admin-controlled wallet.

Patch

Fixed in ed6ecef.

OS-GFX-SUG-01 [resolved] | Curve arithmetic

Description

Consider a swap where token X is being transferred in and token Y is being transferred out. We have the following quantities:

- x and y, the current pool balances of X and Y;
- w_x and w_y, the current weights of X and Y;
- p, the current oracle price of Y in terms of X;
- A, a curve parameter.

In order to determine the price of a swap, the gfx-ssl program uses the following curve:

$$\frac{dy}{dx} = -\frac{y}{x}\frac{w_x}{w_y}\frac{Ap\frac{x}{w_x} + \frac{y}{w_y}}{A\frac{y}{w_y} + p\frac{x}{w_x}}dx,$$

whose closed-form equation is

$$\frac{w_x(A^2-1)\ln\left[w_x^2\frac{y}{x} + Aw_xw_y(p+\frac{y}{x}) + (w_x + Aw_y)\ln\frac{y}{x}\right]}{(Aw_x + w_y)(Aw_y + w_x)} + \frac{\ln x}{w_y} = C.$$

The left-hand side is implemented in a closure to calculate the constant C.

gfx-ssl/src/math/curves.rs RUST

```rust
let curve = |x: Decimal, y: Decimal| {
    (wx0 * (A.powi(2) - Decimal::ONE)
        * (wx0.powi(2) * y / x + wx0 * A * wy0 * (p + y / x) + wy0.powi(2) * p).ln()
        + (wx0 + A * wy0) * (y / x).ln())
        / ((wx0 * A + wy0) * (wx0 + A * wy0))
        + x.ln() / wy0
};
```

However, the program later uses a different closed-form equation to calculate the point where the pool price crosses the oracle price.

gfx-ssl/src/math/curves.rs RUST

```rust
let g = wy0_
    * (A_.powi(2) * C_ * wx0_ * wy0_
        - A_.powi(2) * wx0_ * (p_ * wy0_ * (A_ * wx0_ + A_ * wy0_ + wx0_ + wy0_)).ln()
```

```
        + A_ * C_ * wx0_.powi(2)
        + A_ * C_ * wy0_.powi(2)
        - A_ * wy0_ * (p_ * wy0_ / wx0_).ln()
        + C_ * wx0_ * wy0_
        - wx0_ * (p_ * wy0_ / wx0_).ln()
        + wx0_ * (p_ * wy0_ * (A_ * wx0_ + A_ * wy0_ + wx0_ +
        ↪  wy0_)).ln())
    / ((A_ * wx0_ + wy0_) * (A_ * wy0_ + wx0_));

let x1 = Decimal::from_f64(g.exp()).unwrap();
```

Even though these are equivalent formulations, it would be safer to reuse logic. In the program's current state, modifying the curve would require updating two different closed-form equations. The closure is also more efficient in terms of operation count.

Remediation

Notice that the first term in the closed-form equation only references

$$\frac{y}{x} = p\frac{w_y}{w_x},$$

which is always known. Extract the first term's calculation from the closure, and reuse it when calculating when the pool price crosses the oracle price.

Patch

Fixed in 66cc5a1.

A | Program Files

Below are the files in scope for this audit and their corresponding SHA256 hashes.

gfx-controller

```
Cargo.toml                              9a5a61804745504f312fb659116ba092744456d9280c307a7a56f913286c418c
README.md                               e7064190e1d4a6543eb689f2acb6cd6678d349a4f99ac4be3b898b699e35e882
Xargo.toml                              815f2dfb6197712a703a8e1f75b03c6991721e9eb7c40dfaec8b0b49da4aa629
build.rs                                0fc6584cbb5f0c7479701804e57b4923ecf8a07c959abad3c40a0481c7d8920e
examples
  config_controller.rs                  3b8ccc9c8b7a01d977f1c7f07f17afae34028524b4dc85f3b1c924190d3959eb
  create_controller.rs                  e269213a0b202b3e9acf1748196db786be3b00db74f1b31e20e770fa907a846b
  distribute_staking_reward.rs          6526f34ae025657fcd592c82f69ddbce07e3dd2ce77fd5d028e4add6db6bb05f
  print_vault.rs                        655c52ce2d399d3689e8827898dfdb6ff316bc23f4834d54b28e3f1fcd6f7d0a
  stake.rs                              1825d5c5d8e6476d0eab69b638d618012ba63e22773672985a6ad7b94942ed0a
  suspend.rs                            313bbef7c676e9d965a25226f9455897098a63ea874048310eb4fcd8428f029f
  unstake.rs                            4b848a8fba20fde5b6f13160d515cf58aed0cabf1dfe1ebe3bf615d2e24fa52f
src
  cli.rs                                62946864bbf10fd66eb91cdad37e2f06f9f6573446db496dcb790adc69a3ebb4
  errors.rs                             d96043ffaa750ad9d668ae8e98bef006042adfb758ab88ad7ec3ac6abd1dd4ef
  lib.rs                                263f23dcf4a28a6fd7967798f8a4b3511e1320b66f0c410f86f41c33c365a16b
  utils.rs                              2d2db588959d3868099955d98778a0244a9108ad7b3b0cbe0fcf3a02ee236295
  contexts
    config_controller.rs                cb07569be65b850feb58c8e0f199e4daebc9c981aa9fef2e63a0b88c2ea6310e
    create_controller.rs                97c8e65178097080473840a452141e5ab8c3514de5f74ca1954893676ebf0821
    create_staking_account.rs           f1a694d58f92e8aa998ae58011ae1c8c9ba3e736d7245793737fd0d905f0b5c0
    distribute_staking_reward.rs        fd2429102779767e7d6ae30a14d097131cc930f5aa312ef68f4c79944ba85259
    mod.rs                              d726e12f9d9b438f095e7163aa1ae70711b8daac59270707fb7c65b1aa94c592
    stake.rs                            ec3bcb8db0040bd3633bd1af02560f795e2d147eb91c8817788bc981d1e524c9
    unstake.rs                          c950ec9fa22d479caa7c2decd1d3fd5a94ab1cd93a89f3b053b8444505a52a4c
  states
    controller.rs                       ab74ae15e649a04c66c09233cf81a7e69bebf95a647f99302381fe473d3653d9
    mod.rs                              3fec5e159bc954c59cfee0f45e3faf0a7f4dfbdc194e337b2aff150c64e83b24
    staking_account.rs                  41811f644e525aa4917933509556a886069badb8d6a1605461dd02e7e26d3261
tests
  create_controller.rs                  f7e48bd0fcca244410e1222de3cfce6017de72ee670f275f20b74860a9c5292c
  create_staking_account.rs             6a81f22e0ba058b9f08f113381083617b96e90e90b710568c3f6467a029b5dd0
  dividend.rs                           58d6dde1114d02ae045abec66675fb071d9d1ad369936c5149ebada9b36b39ac
  stake.rs                              7f4734ac291f49c468ceb736638a0bfeba33f0f9926fdf9685f98b397c0538ee
  unstake.rs                            3e0198b2697469d1034aa9f1156a6c8cceddc2b752e801e78c9f122cabbe1292
```

gfx-ssl

```
Cargo.toml                              60ea212ffd717c5906df2927ceb27649d3fd6096321c04c38e65d71d29b5997f
README.md                               09dde43d2ab1bcfddc402c46b6d90279a285438a8e08f00b55e15dd7b568ec43
Xargo.toml                              815f2dfb6197712a703a8e1f75b03c6991721e9eb7c40dfaec8b0b49da4aa629
build.rs                                0fc6584cbb5f0c7479701804e57b4923ecf8a07c959abad3c40a0481c7d8920e
examples
  config_pair.rs                        59b9cfc8719b432501d0fb410f9a84759b73262e5daabeacf5afc5624e7d2708
  config_ssl.rs                         9f7941d376459e6505040a5e85201ae538116ecced85d0c505a43e5fc9a021dd
  create_pair.rs                        d0e193642bb5a1978feb9f2daf4eabf279f287c3bc48a57d2945f9e83a8bddc8
  create_ssl.rs                         7cecb0918e2a93dbdc9e80ca1fbab61dd833819428a950ffde9e984e5ce5bba0
  create_swapped_liability_vault.rs     70077bd79e662244010f79d8741b8fd19b4450c565aee036b6239e06f294c291
  deposit.rs                            2609506bb8b9768d6a5acad2d0c9a8d7efff2a5628c572e0c35c7f2b7000f182
  inspect.rs                            6b8a3d46daedc5f675b95c5edda7f6c81cdbc0f20d663a857b4be196b0383629
  layout.rs                             89d3334ba3296cbbdf6da5b7164652863db5996ef52b89972d332897f222668e
  replace_oracle.rs                     52722bae0e362f32c60e84a9d9615da6bb28eec940c6e7a47d391578bda0c19d
  swap.rs                               b67f12328223ba9dbc8eb096a59f4a0646f815c1f4fb73f1f18685c8b54d7fb5
  withdraw.rs                           b008f2377838a77b3f91cea5bec255d4a4ee7a1513c9dc992a547edf753b8d7c
src
  cli.rs                                a340c917c09c43a740e90100b5cf198b38a54438c8d09de2068cdfe095daf14d
  errors.rs                             18c6ee2cd12b9e909ace3e1d47f3c4c7b4042a45489081d8f9759014c481366c
  lib.rs                                950e0e5102c43e61893905199590c9912c036619fe833c7f3481121a7607d3df
  svec.rs                               e9256d82477353001a87e9869b0b1e97526b67a95081c9c498159552048e4083
  utils.rs                              591a597415739f576fec6d28122cfecc3c7c7b9def67ca04c9cfe5f78a37eb0a
```



```
  contexts
    mod.rs                                                           931ad39076398c6c966593fa59f3aff4d7a7055c8e61005d8f94e7fef801408b
    permissionless
      burn_pt.rs                                                     97de8cc44723e9467c83b2362c219a56c594d532fa4ed3aefeefbda277e6fbfd
      create_liquidity_account.rs                                    f455a21c154ed8f8806695e80099388ef6d1d419f0cc755a84341ce35471efdb
      deposit.rs                                                     3c7e545c8a0c8fd26bd19687c2a8cd7c1c79b20e71093fd0acfd978380d3809a
      mint_pt.rs                                                     9972c18e70549b5f2af39cc2bcb9ece96a50f75565edf2eb1a4007ccd1d23169
      mod.rs                                                         326f162ba94b140505d84a61df1afe43853ed8a271574451beb993f463642104
      swap.rs                                                        d6f676e991cc1b2cc79474af2ab814c4694f349bce1c25f8fd2f7b1698538a57
      withdraw.rs                                                    cbdf05a3314a724cfc6d1af5e9460799241c77d45374ca099b745ea085f25f82
    privileged
      config_pair.rs                                                 ce9f4122aa1fc2db3239b02dfc85fdbb6ff2320120ba1103ef8928575230d752
      config_ssl.rs                                                  ed8987acf6ffd5939012ca13eddfd3c8c9174d74eb6520ea363bc5fd8a2ef8b8
      crank_liability.rs                                             c495ee3c58745a31e18fee1ad83bfd7d0ff23c2ebeaa851626a3c258e923552b
      create_pair.rs                                                 c0ed8076e717149e59f3642a241488faa0dd91448e61a1d1c3b281a0846873fc
      create_ssl.rs                                                  f78913b081f7a1836017704641cdcaa47fc5a193f746401e7630ec5dd67b4b75
      mod.rs                                                         1abfc5be7e979fb9e890b74e21fb6886157d4571363c0eabd0d543af3d91ae81
  math
    curves.rs                                                        4d2cead5f45355f78cf51bf94cc79d6e59f4c8d8dddf18e0e7d2df89dff95fd1
    linalg.rs                                                        68ea9c9b93b6c72588c5fce533fb4889b0ebacbb9a370470f0ef3e76b3f0afd8
    mod.rs                                                           c8d1369423e0bef024bd4969c6624a32efc73448f0b208d9d5f90e03f2a01eba
    rk45.rs                                                          dd9a98831c5ad750d79dccdbedb7dd570f7a7c6c844f8da992a93caf622dc176
  python
    constant_product.rs                                              7f02c3834f56a0a6ad5401b958eccdfe7123a8493e28f5e4e17ef438a20ffa8c
    errors.rs                                                        af9c087418ecebbeb2bfd5d31a6156fc9a7b7acb74891a2e11f5ea1a91ff9805
    mod.rs                                                           5aa2de8fe66e332d86fb1d7330dfa5a5c34bb45c2718ac904ee7ea10421f0477
    pair.rs                                                          16cb77ab0ee1f06b0a2fdf7f6536ae04590e97701d9e2b05a4f680a1cbd578aa
    solana.rs                                                        8d2f2bbe40893f33fd04dd0786b73485e8b7e0dc69cdb5218b0bcfbf8457620b
    ssl.rs                                                           5bb3c7170ff328188e735c096cd4021e7d2c546c7b3b6dd639459ae9cfeaccec
  states
    liquidity_account.rs                                             9438528fbe7ee1f1159c071c3629e10d8f82ae784b3d7d09b90e230fcc6a09bf
    mod.rs                                                           9583e783c569ac2de0e8f5f391cebf9a71ccd1a9d7901d72f5a6b39afd4a4990
    pt_mint.rs                                                       b2a1b05ebb2cf38fb14be56828bec3688856c4122681a186494602d68a0a1bb4
    pair
      mod.rs                                                         3c138d56b7b82936c6e3cdc7f15d0afd907f8eeab3e403620e126be1e5d9e46d
      swap.rs                                                        2e2692214f0ec811a460e1b11b71f00714ba047e7692e50db18faffb9ea0ca67
    ssl
      deposit.rs                                                     da0335bf4358c428edcc646d184574931f3904037ce8eec6c32664351731aee5
      mod.rs                                                         a5d9bb5940f4e905f397c3f0c34b266de408312c13419329f104155151c097f9
      withdraw.rs                                                    7f0b7e268784a2df7f2783c304cc0653df80716e002a7c4ff3292526a2cc3019
tests
  create_liquidity_account.rs                                        80840c503c8e09b610c168d3881a6e56cb3e114d3ee70e0c5e3cb37b40cc40cb
  create_pair.rs                                                     828c152ccde102a84e988f9768a7eb8e1859f610061e7c0d8813451913b5c5b5
  create_ssl.rs                                                      f502c475fde95a1bab2b464d4759edc566c2d79ef8100795a43e40795fd8cd2c
  deposit.rs                                                         b0eabf09138a7b3c640b50f0f8a8ec46cecf7e12b9da1651e75b770d0ba735af
  swap.rs                                                            20eccc5c3d91924a7d9c41cb5f7d569a28593f77c284d27894818ccd3ae8c67c
  withdraw.rs                                                        e20e0771782f7d8bee39c3f578c10206a6a24b90432e25eb28458bf4089b0caa
```

B | Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an onchain program. In other words, there is no way to steal tokens or deny service, ignoring any Solana specific quirks such as account ownership issues. An example of a design vulnerability would be an onchain oracle which could be manipulated by flash loans or large deposits.

On the other hand, auditing the implementation of the program requires a deep understanding of Solana's execution model. Some common implementation vulnerabilities include account ownership issues, arithmetic overflows, and rounding bugs. For a non-exhaustive list of security issues we check for, see Appendix C.

Implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to get a comprehensive understanding of the program first. In our audits, we always approach any target in a team of two. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.

C | Implementation Security Checklist

Unsafe arithmetic

Integer underflows or overflows	Unconstrained input sizes could lead to integer over or underflows, causing potentially unexpected behavior. Ensure that for unchecked arithmetic, all integers are properly bounded.
Rounding	Rounding should always be done against the user to avoid potentially exploitable off-by-one vulnerabilities.
Conversions	Rust as conversions can cause truncation if the source value does not fit into the destination type. While this is not undefined behavior, such truncation could still lead to unexpected behavior by the program.

Account security

Account Ownership	Account ownership should be properly checked to avoid type confusion attacks. For Anchor, the safety of unchecked accounts should be clearly justified and immediately obvious.
Accounts	For non-Anchor programs, the type of the account should be explicitly validated to avoid type confusion attacks.
Signer Checks	Privileged operations should ensure that the operation is signed by the correct accounts.
PDA Seeds	PDA seeds are uniquely chosen to differentiate between different object classes, avoiding collision.

Input validation

Timestamps	Timestamp inputs should be properly validated against the current clock time. Timestamps which are meant to be in the future should be explicitly validated so.
Numbers	Sane limits should be put on numerical input data to mitigate the risk of unexpected over and underflows. Input data should be constrained to the smallest size type possible, and upcasted for unchecked arithmetic.
Strings	Strings should have sane size restrictions to prevent denial of service conditions
Internal State	If there is internal state, ensure that there is explicit validation on the input account's state before engaging in any state transitions. For example, only open accounts should be eligible for closing.

Miscellaneous

Libraries	Out of date libraries should not include any publicly disclosed vulnerabilities
Clippy	cargo clippy is an effective linter to detect potential anti-patterns.

D | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the General Findings section.

Critical	Vulnerabilities which immediately lead to loss of user funds with minimal preconditions Examples: • Misconfigured authority/token account validation • Rounding errors on token transfers
High	Vulnerabilities which could lead to loss of user funds but are potentially difficult to exploit. Examples: • Loss of funds requiring specific victim interactions • Exploitation involving high capital requirement with respect to payout
Medium	Vulnerabilities which could lead to denial of service scenarios or degraded usability. Examples: • Malicious input cause computation limit exhaustion • Forced exceptions preventing normal use
Low	Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions
Informational	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants • Improved input validation • Uncaught Rust errors (vector out of bounds indexing)