Socean

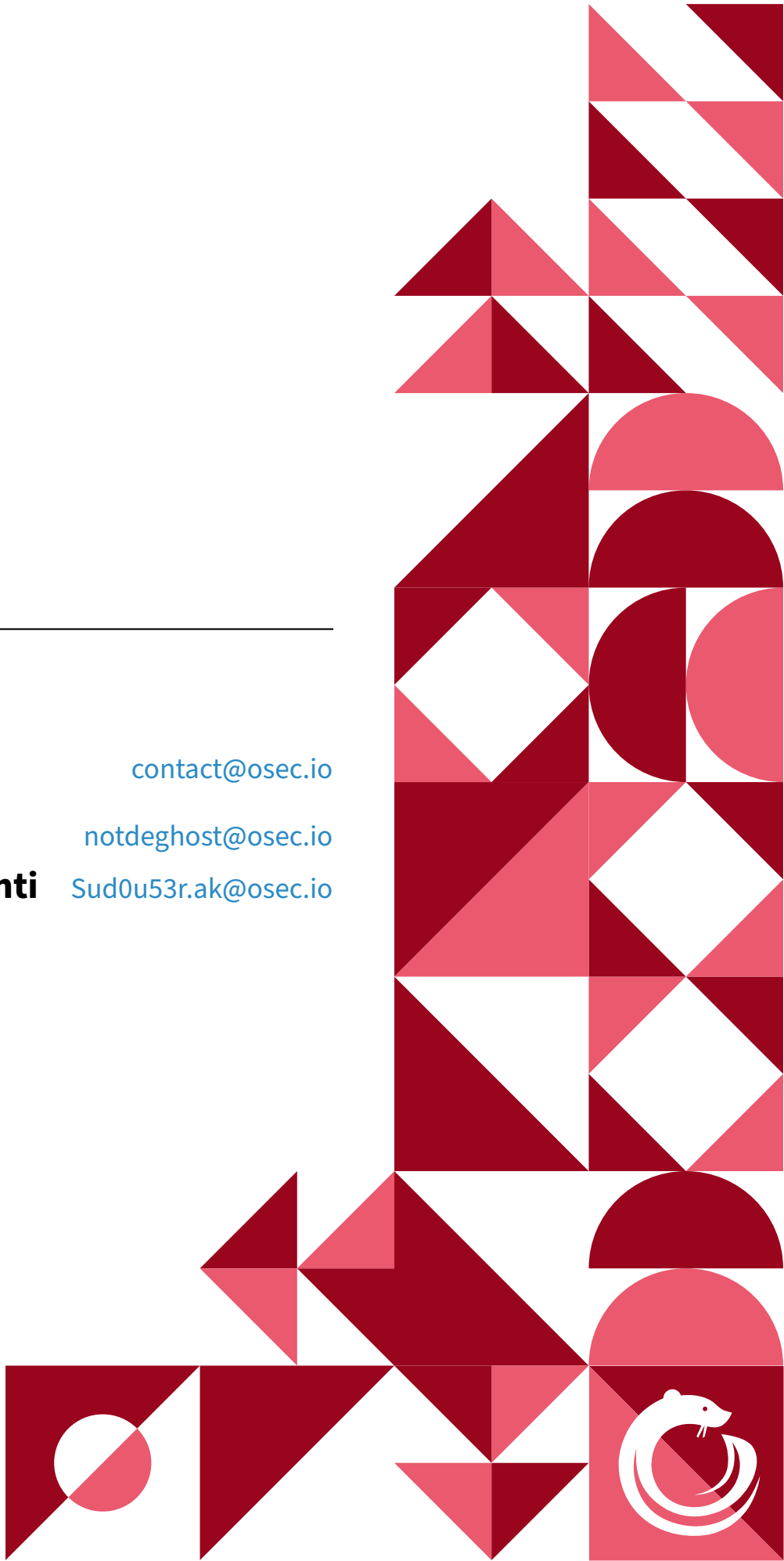
Audit

Presented by:

OtterSec contact@osec.io

Robert Chen notdeghost@osec.io

Akash Gurugunti Sud0u53r.ak@osec.io



Contents

Appendices

01 | Executive Summary

Overview

Socean engaged OtterSec to perform an assessment of the `descending-auction-program` and `bonding` programs.

This assessment was conducted between May 30th and June 8th, 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 **[not yet delivered]**.

Key Findings

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

- 4 findings total
- No loss of funds issues

As part of this audit, we also provided proofs of concept for each vulnerability to prove exploitability and enable simple regression testing. These scripts can be found at https://osec.io/pocs/socean-pocs. For a full list, see Appendix B.

02 | Scope

The source code was delivered to us in a git repository at github.com/igneous-labs/descending-auction-program and github.com/igneous-labs/bonding. This audit was performed against commits `3e1deb6` and `439c88d` respectively.

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
descending-auction-program	An auction that descends in price over time.
bonding	Token vesting with underlying bonded tokens.

03 | 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.

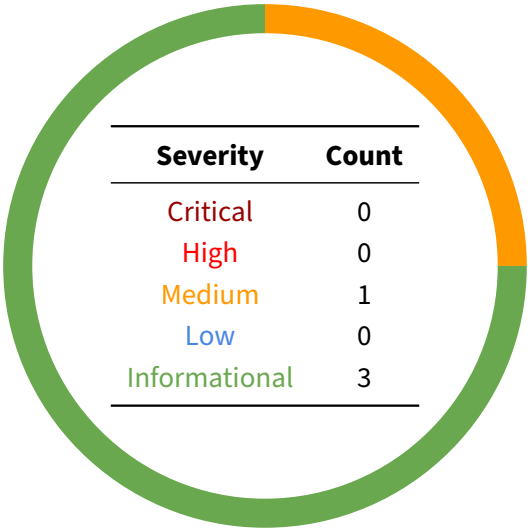
While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.

04 | Findings

Overall, we report 4 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	0
Medium	1
Low	0
Informational	3

Proof of Concepts

For each vulnerability we created a proof of concept to enable easy regression testing. We recommend integrating these as part of a comprehensive test suite. The proof of concept directory structure can be found in Appendix B.

A GitHub repository containing these proofs of concept can be found at https://osec.io/pocs/socean-pocs.

To run a POC:

```sh
./run.sh <directory name>
```

For example,

```sh
./run.sh os-soc-adv-00
```

Each proof of concept comes with its own patch file which modifies the existing test framework to demonstrate the relevant vulnerability. We also recommend integrating these patches into the test suite to prevent regressions.

05 | 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-SOC-ADV-00	Medium	TODO	Payment destination account controlled by the auction authority can give author power to close auction while it is still in progress

OS-SOC-ADV-00 [med] [TODO] | Auction authority controls auction in unintended way

Description

In the `InitializeAuction` instruction, the authority of the auction sets the `payment_destination` account for the auction to receive payments.

Since this account is controlled by the authority, the authority may close this account, temporarily closing the auction by failing the subsequent transfer in the Purchase instruction.

Proof of Concept

Consider the following scenario:

1. Create an auction and set the `payment_destination` token account that can be controlled by the authority.
2. Now, while the auction state is InProgress, close the `payment_destination` token account.
3. Try to call the `Purchase`instruction as a user trying to purchase some sale tokens.

The `Purchase`instruction will fail, since payment tokens cannot be sent to a closed account.

Remediation

Possible ways to avoid the above mentioned scenario:

1. The `payment_destination` account can be set as a PDA generated with `auction.key()` as seed. Then another instruction can be written that checks the authority of the auction and lets him transfer the tokens from `payment_destination` account to specified account.
2. The `payment_destination` account can be set as a PDA generated with `auction.key()` as seed, and then delegate `u64::MAX` amount to authority specified account using the `Approve`instruction.

Patch

06 | 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	Description
OS-SOC-SUG-00	Using `saturating_sub` instead of writing an if condition in auction program
OS-SOC-SUG-01	Unnecessary account input to DepositToAuctionPool ix in auction program
OS-SOC-SUG-02	Unnecessary explicit specification of `owner = id()` in Initialize and Vest ix in bonding program

OS-SOC-SUG-00 | Use `saturating_sub` instead

Description

In Purchase instruction, at line 120, the value of `t` is calculated as `t - dt`, if `t > dt`, otherwise `0`. This can be rewritten using `saturating_sub`.

descending-auction-program/src/instructions/purchase.rs RUST

```rust
let t = if t > dt {
    t.checked_sub(dt)
} else {
    Some(0)
}.ok_or(AuctionError::InternalError)?;
```

Remediation

The above code can be rewritten as follows using the `saturating_sub` method:

descending-auction-program/src/instructions/purchase.rs RUST

```rust
let t = t.saturating_sub(dt);
```

OS-SOC-SUG-01 | Account unnecessarily taken as input

Description

In DepositToAuctionPool ix, the auction_authority account is taken as input but not used anywhere in the run function. It is used only to check the auction_pool.owner, which is an unnecessary constraint, because auction_pool is a PDA generated using auction.key() and auction.sale_mint as seeds, so it will be unique to an auction just like auction_authority. Since the owner of the auction_pool is only set in InitializeAuction ix, the owner check here is unnecessary.

Remediation

The constraint in the auction_pool checking for the owner of the auction_pool can be removed and the auction_authority account can be removed from the DepositToAuctionPool ix as well.

OS-SOC-SUG-02 | `owner = id()` is not necessary

Description

For bond_pool account in Initialize ix and for vesting account in Vest ix, the owner is explicitly set to id().

bonding/src/instructions/initialize_bond_pool.rs RUST

```rust
pub struct Initialize<'info> {
    // ...
    #[account(
        init,
        payer = owner,
        space = BondPool::LEN,
        owner = id()
    )]
    pub bond_pool: Account<'info, BondPool>,
    // ...
}
```

bonding/src/instructions/vest.rs RUST

```rust
pub struct Vest<'info> {
    // ...
    #[account(
        init,
        payer = payer,
        space = Vesting::LEN,
        owner = id(),
        seeds = [
            BONDING_PREFIX,
            VESTING_PREFIX,
            &bond_pool.key().to_bytes(),
            &user.key().to_bytes(),
            &[seed]
        ],
        bump,
    )]
    pub vesting: Account<'info, Vesting>,
    // ...
}
```

Remediation

The ”owner = id()” can be removed, because the owner of the structs defined in the program will be set to program_id by the anchor.

A | Program Files

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

```
descending-auction-program/
  Cargo.toml                                  30e522ae6a258ae84c6a5b0398335fff
  Xargo.toml                                  815f2dfb6197712a703a8e1f75b03c69
  src
    curves.rs                                 b657041d55beb15a8927baa47a9205e9
    errors.rs                                 7064d4e9005cbe520bbaf06f9dc885fb
    lib.rs                                    9461bfc1f6baf1b7e24eea14e0c4ed86
    state.rs                                  65255167c3974cead7621df00c3eeb40
    instructions
      close_auction.rs                        1a12ef7891d15bb0a6447aba8d31f0f5
      deposit_to_auction_pool.rs              67136050edf226a2422b37a895fd394a
      initialize_auction.rs                   e8f9e39ae78eeefd35b78f86b0966cd1
      mod.rs                                  6e80a4c685ff29a4fc911901ec81b818
      purchase.rs                             19a4f12d1bd9ca71658f45e70878bf01
      update_authority.rs                     3c5ce8ab581c0c9034463621b8be202f
      update_ceil_price.rs                    b32db34f468e81a9317cdd96e55f92fd
      update_end_time.rs                      4b6e19eb373750d1cb4b892c9e1d3931
      update_floor_price.rs                   839de0d9daa9253b8f50f47c985c8dc8
      update_start_time.rs                    b398c8a976d05e3c1bc9603ea5191a95


bonding/
  Cargo.toml                                  a71316c342593540ac2d13708b062a77
  Xargo.toml                                  815f2dfb6197712a703a8e1f75b03c69
  src
    errors.rs                                 0766d6cc0e74cde2cd193b47b51f7a89
    lib.rs                                    2646ec346a39a951f99d5206cf9a3f27
    validators.rs                             e972c4696285e5fc571357397d7381eb
    instructions
      cancel_vest.rs                          8d40b604891d0e1095be833a922042f5
      claim.rs                                327544f7af03cd65640f749265c355c9
      deposit.rs                              01c1ff2da23a1b0d463923715c905247
      initialize_bond_pool.rs                 39c871306ccfb9f6717233cfb77b2598
      mod.rs                                  3c26e47f77c93fbac4db6b7cd4344632
      set_owner.rs                            48166cc3a23d1900fbc532425cb263cc
      set_vesting_seconds.rs                  89c5f9d20c976a3381eb934c23edfa95
      vest.rs                                 3ec271ee34f1efbaf2da4b0425ccc527
    state
      bond_pool.rs                            c6a437e1f3a0b32b9f65318d04ba8c68
      mod.rs                                  524e733168436b78d63da0ae1ea311ad
      vesting.rs                              bb21270650dbefd0e631c1441e97f731
```

B | Proof of Concepts

Below are the provided proof of concept files and their corresponding SHA256 hashes.

```
os-soc-adv-00
  hash                                                e6e84f47f846889a1a6fba17f22b87fc
  patch                                               e0ff075baa268ceb2118fd25ce638126
  run.sh                                              04ffe52fa5854c3a44a27223670abe29
```

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)