Convergence

Audit

Presented by:

OtterSec contact@osec.io

Robert Chen notdeghost@osec.io

Akash Gurugunti Sud0u53r.ak@osec.io

Contents

Appendices

01 | Executive Summary

Overview

Convergence engaged OtterSec to perform an assessment of the `convergence-rfq` program.

This assessment was conducted between 0th and 0th, 0.

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.

- 8 findings total
- 2 vulnerabilities which could lead to loss of funds
  - OS-CVG-ADV-00: Tokens locked in escrow account forever
  - **??**: Reduces fee due to miscalculation

As part of this audit, we also provided proof of concepts for each vulnerability to prove exploitability and enable simple regression testing. These scripts can be found at github.com/otter-sec/convergence-rfq-rfq-pocs. For a full list, see Appendix B.

02 | Scope

The source code was delivered to us in a git repository at github.com/convergence-rfq/rfq. This audit was performed against commit 9a64a06.

There was a total of one program 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.

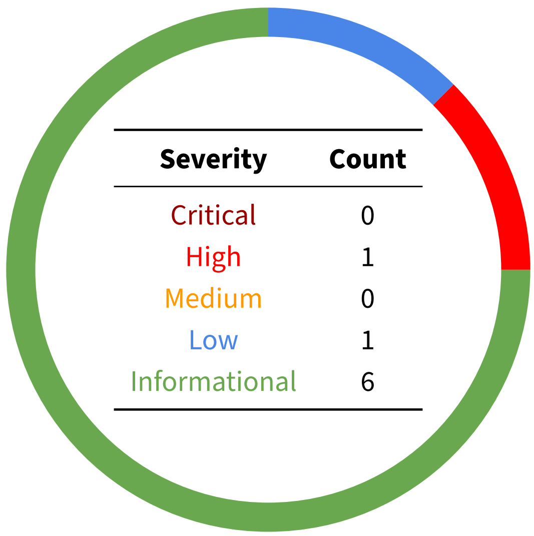
Name	Description
rfq	A DeFi primitive for over the counter transactions with no price slippage.

03 | Findings

Overall, we report 8 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	0
Low	1
Informational	6

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 proof of concepts can be found at osec.io/pocs/convergence-rfq.

To run a POC:

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

For example,

```sh
./run.sh os-cvg-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.

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 E.

ID	Severity	Status	Description
OS-CVG-ADV-00	High	TODO	Cancel instruction works even after the RFQ is confirmed that enables taker to lock maker tokens in escrow account.
OS-CVG-ADV-01	Low	TODO	CPI call being made to unchecked program address.

OS-CVG-ADV-00 [high] | Cancel Instruction Works on Confirmed RFQ

Description

An RFQ can be cancelled by the taker even after the RFQ is confirmed. This enables the taker to confirm an order, settle the order using the `Settle` instruction and then cancel the RFQ using `Cancel` instruction, which prevents the maker from settling their order, because `Settle` instruction prevents settling for cancelled RFQs. This results in permanent locking of maker funds in the escrow.

The code snippet below shows the access_control used for the cancel instruction which only checks if signer is the taker of the RFQ and if the RFQ is not cancelled or expired yet.

rfq/src/access_control.rs RUST

```rust
/// Cancel access control.
///
/// Ensures:
/// - RFQ belongs to signer
/// - RFQ not canceled
/// - RFQ not expired
/// - RFQ has no responses
pub fn cancel_access_control<'info>(ctx: &Context<Cancel<'info>>) ->
    ↪ Result<()> {
    let signer = ctx.accounts.signer.key();
    let rfq = &ctx.accounts.rfq;

    let authority = rfq.authority.key();

    require!(authority == signer, ProtocolError::InvalidAuthority);
    require!(!rfq.canceled, ProtocolError::InvalidCancel);
    require!(
        Clock::get().unwrap().unix_timestamp < rfq.expiry,
        ProtocolError::RfqInactive
    );

    Ok(())
}
```

Proof of Concept

Consider the following scenario:

1. A taker creates an RFQ using `Request` instruction.

2. A maker then responds to that RFQ using `Respond` instruction.
3. The taker then confirms, settles and cancels the RFQ using `Confirm`, `Settle` and `Cancel` instructions.
4. Now, if the maker tries to settle the RFQ using `Settle`, the transaction fails because cancelled RFQs cannot be settled.

Remediation

Add a constraint to `cancel_access_control` that prevents a taker from cancelling a confirmed RFQ.

rfq/src/access_control.rs DIFF

```diff
  pub fn cancel_access_control<'info>(ctx: &Context<Cancel<'info>>) -> Result<()> {
      let signer = ctx.accounts.signer.key();
      let rfq = &ctx.accounts.rfq;

      let authority = rfq.authority.key();

      require!(authority == signer, ProtocolError::InvalidAuthority);
      require!(!rfq.canceled, ProtocolError::InvalidCancel);
+     require!(!rfq.confirmed, ProtocolError::InvalidCancel);
      require!(
          Clock::get().unwrap().unix_timestamp < rfq.expiry,
          ProtocolError::RfqInactive
      );

      Ok(())
  }
```

Patch

OS-CVG-ADV-01 [low] | Unchecked psy_american_program Address

Description

In `InitializeAmericanOptionMarket` and `MintAmericanOption` instructions, the address that is passed to `psy_american_program` account is not being validated against any address. Making a CPI call to arbitrary address can lead to unintended behaviors in the program.

In the below code snippets from the psyoptions, the `psy_american_program` account is not checked against any predefined and any arbitrary program address can be passed to it.

rfq/src/psyoptions/context.rs RUST

```rust
#[derive(Accounts)]
pub struct InitializeAmericanOptionMarket<'info> {
    #[account(mut)]
    pub user: Signer<'info>,
    /// CHECK: TODO
    pub psy_american_program: AccountInfo<'info>,
    pub underlying_asset_mint: Box<Account<'info, Mint>>,
```

rfq/src/psyoptions/context.rs RUST

```rust
#[derive(Accounts)]
pub struct MintAmericanOption<'info> {
    #[account(mut)]
    pub authority: Signer<'info>,
    /// CHECK: TODO
    pub psy_american_program: AccountInfo<'info>,
    /// The vault where the underlying assets are held. This is the PsyAmerican
    #[account(mut)]
    pub vault: Box<Account<'info, TokenAccount>>,
```

Proof of Concept

Consider the following scenario:

1. A malicious user calls `MintAmericanOption` instruction with `psy_american_program` set to our own program address that exits successfully when called.
2. He can set the `leg.processed = true` for any RFQ without even minting options tokens from the psy_options market.

Remediation

Add a constraint to check the address of the `psy_american_program` account against a predefined address.

Patch

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-CVG-SUG-00	TODO	Improper formula used for fee calculation leads to significant reduction of fee amount.
OS-CVG-SUG-01	TODO	Unnecessary owner checks for state accounts.
OS-CVG-SUG-02	TODO	Unnecessary treasury wallet field in protocol state.
OS-CVG-SUG-03	TODO	A constraint to check if fee_numerator < fee_denominator in the protocol state.
OS-CVG-SUG-04	TODO	Cancel instruction doesn't implement constraint that checks if an RFQ has responses as mentioned in the docstring.
OS-CVG-SUG-05	TODO	Confirm instruction allows taker to confirm cancelled RFQs.

OS-CVG-SUG-00 | Reduced Fee Due To Miscalculation

Description

In the `Settle` instruction, the fee_amount is calculated as

rfq/src/instructions.rs RUST

```rust
    fee_amount = (rfq.order_amount as u128)
        .checked_div(protocol.fee_denominator as u128)
        .ok_or(ProtocolError::Math)?
        .checked_mul(protocol.fee_numerator as u128)
        .ok_or(ProtocolError::Math)?
        .to_u64()
        .ok_or(ProtocolError::Math)?;
```

There are also some other instances where the fee_amount is calculated similarly. The problem with this is that the fee_amount can be significantly lower than the precisely calculated value based on the numerator and denominator values. For example,

let rfq.order_amount = 99
let fee_numerator = 15
let fee_denominator = 100
fee_amount = (99 / 100) * 15 = 0

While the precise fee_amount = 14.85. By doing multiplication before division will give more precise value fee_amount = (99 * 15) / 100 = 14

Remediation

Change the fee calculation to do the multiplication before division.

OS-CVG-SUG-01 | Unnecessary Owner Checks for State Accounts

Description

The accounts in the instructions that are specified using the syntax `Account<'info, AccountState>` or `Box<Account<'info, AccountState>>` need not have a constraint that checks its owner, i.e., `account.to_account_info().owner == program_id`. The anchor framework implicitly checks if the owner of the account is equal to the program address in which the AccountState struct is defined.

A sample of the affected code can be found in the snippet below.

rfq/src/contexts.rs RUST

```rust
        seeds = [PROTOCOL_SEED.as_bytes()],
        bump = protocol.bump,
        constraint = protocol.to_account_info().owner == program_id
    )]
    pub protocol: Account<'info, ProtocolState>,
```

Remediation

Remove the unnecessary constraints on the state accounts.

OS-CVG-SUG-02 | Unnecessary Treasury Wallet Field in Protocol State

Description

The account to which the fee amount is transferred in the `Settle` instruction should be of the same mint as the `asset_escrow` or the `quote_escrow` based on the caller of the instruction. Since there can be no fixed token account for collecting the fee, the treasury wallet token account field in the protocol state is unusable.

The highlighted lines in the below code snippet can be removed.

rfq/src/state.rs RUST

```rust
/// Protocol state.
#[account]
pub struct ProtocolState {
    // Protocol authority
    pub authority: Pubkey,
    // PDA bump
    pub bump: u8,
    // Fee denominator
    pub fee_denominator: u64,
    // Fee numerator
    pub fee_numerator: u64,
    // Treasury wallet
    pub treasury_wallet: Pubkey,
}
```

Remediation

Remove the unnecessary treasury wallet field in the protocol state.

OS-CVG-SUG-03 | Missing Check During Fee Initialization

Description

The fee_numerator should be less than the fee_denominator in the protocol state. If the fee_numerator is greater than the fee_denominator, then while calculating fee_amount in the `Settle` instruction, the `rfq.order_amount` or the `rfq.best_bid_amount` is divided by fee_denominator and multiplied with fee_numerator resulting in fee_amount greater than the rfq.order_amount or rfq.best_bid_amount.

This results in the instruction fails while executing checked_sub that subtracts fee_amount from the order_amount or best_bid_amount and renders the protocol unusable until it is changed again.

Remediation

Add a constraint in `initialize_access_control` and `set_fee_access_control` access control functions to check that `fee_numerator < fee_denominator`.

rfq/src/access_control.rs DIFF

```diff
pub fn initialize_access_control<'info>(
    _ctx: &Context<Initialize<'info>>,
    fee_denominator: u64,
) -> Result<()> {
    require!(fee_denominator > 0, ProtocolError::InvalidFee);
+   require!(fee_numerator < fee_denominator, ProtocolError::InvalidFee);
    Ok(())
}
```

rfq/src/access_control.rs DIFF

```diff
pub fn set_fee_access_control<'info>(
    ctx: &Context<SetFee<'info>>,
    fee_denominator: u64,
) -> Result<()> {
    let signer = ctx.accounts.signer.key();
    let authority = ctx.accounts.protocol.authority.key();

    require!(signer == authority, ProtocolError::InvalidAuthority);
    require!(fee_denominator > 0, ProtocolError::InvalidFee);
+   require!(fee_numerator < fee_denominator, ProtocolError::InvalidFee);
```

```
    Ok(())
}
```


OS-CVG-SUG-04 | Taker Can Cancel RFQs With Responses

Description

In access control function for the `Cancel` instruction, it is not checking whether the RFQ has any responses or not as mentioned in the docstring above the `cancel_access_control` (highlighted part in the below code snippet). This allows a taker to cancel an RFQ which has responses.

rfq/src/access_control.rs RUST

```rust
/// - RFQ not canceled
/// - RFQ not expired
/// - RFQ has no responses
pub fn cancel_access_control<'info>(ctx: &Context<Cancel<'info>>) -> Result<()> {
    let signer = ctx.accounts.signer.key();
```

Remediation

Add a constraint to check if the RFQ has responses.

rfq/src/access_control.rs DIFF

```diff
pub fn cancel_access_control<'info>(ctx: &Context<Cancel<'info>>) -> Result<()> {
    let signer = ctx.accounts.signer.key();
    let rfq = &ctx.accounts.rfq;

    let authority = rfq.authority.key();

    require!(authority == signer, ProtocolError::InvalidAuthority);
    require!(!rfq.canceled, ProtocolError::InvalidCancel);
+   require!(rfq.best_bid_amount.is_none() && rfq.best_ask_amount.is_none(), ProtocolError::InvalidCancel);
    require!(
        Clock::get().unwrap().unix_timestamp < rfq.expiry,
        ProtocolError::RfqInactive
    );

    Ok(())
}
```

OS-CVG-SUG-05 | Taker Can Confirm Cancelled RFQs

Description

In access control function for the `Confirm` instruction, it is not checking whether the RFQ is cancelled or not. This allows a taker to confirm a cancelled RFQ, which leads to locking of funds of both taker and maker since `Settle` instruction doesn't allow them to withdraw their funds from a cancelled RFQ.

It can be seen in the code snippet below, that the program does not check if the RFQ was cancelled before confirming the order.

rfq/src/access_control.rs RUST

```rust
pub fn confirm_access_control<'info>(ctx: &Context<Confirm<'info>>,
    ↪ quote: Quote) -> Result<()> {
    let order = &ctx.accounts.order;
    let rfq = &ctx.accounts.rfq;

    let taker = rfq.authority.key();
    let signer = ctx.accounts.signer.key();

    require!(rfq.key() == order.rfq.key(), ProtocolError::InvalidRfq);
    require!(taker == signer, ProtocolError::InvalidTaker);
    require!(!rfq.confirmed, ProtocolError::RfqConfirmed);
    require!(
        rfq.expiry > Clock::get().unwrap().unix_timestamp,
        ProtocolError::RfqInactive
    );
}
```

Remediation

Add a constraint in `confirm_access_control` to check if the RFQ is cancelled or not.

rfq/src/access_control.rs DIFF

```diff
pub fn confirm_access_control<'info>(ctx: &Context<Confirm<'info>>,
    ↪ quote: Quote) -> Result<()> {
    let order = &ctx.accounts.order;
    let rfq = &ctx.accounts.rfq;

    let taker = rfq.authority.key();
```



```
    let signer = ctx.accounts.signer.key();

    require!(rfq.key() == order.rfq.key(), ProtocolError::InvalidRfq);
    require!(taker == signer, ProtocolError::InvalidTaker);
    require!(!rfq.confirmed, ProtocolError::RfqConfirmed);
+   require!(!rfq.canceled, ProtocolError::RfqCanceled);
    require!(
        rfq.expiry > Clock::get().unwrap().unix_timestamp,
        ProtocolError::RfqInactive
    );
```

A | Program Files

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

```
rfq/
  Cargo.toml                      99308fd40abd483616ebd9ac67c51f04a549a1f6c0d6811bb24d1d0c27290089
  Xargo.toml                      815f2dfb6197712a703a8e1f75b03c6991721e9eb7c40dfaec8b0b49da4aa629
  src/
    access_controls.rs            7b22f351cdeda6f9bf496fecdbd3919f4767264b51bef8e593ebfa9e3dcf5a6c
    constants.rs                  cb4dd009b7579dfa5745da89aca2b7685780dccc2c6d048d63bac1cb46ed30cd
    contexts.rs                   709a654d0fa8100fb207802b3129e6cb6ab7d7d345d3fe43a34ebe8de8c01dad
    errors.rs                     684ffb998519242a4ea9bc4f9da9b088369be0b1096245c9d4bbb6e91e61abc0
    instructions.rs               bf13fdfafe35c6eef4c362f2d02c51b501242b0605a392bc3833c2f7cf6ccacb
    lib.rs                        9668b15e1b3afc9b425c3b003c645ddf348fb39d246e17f942612d06fda46c72
    states.rs                     808950ac95d51f8bf5fe9f0633082ac9f506ad17954f3abd9ad214272cb50e5b
    psyoptions/
      contexts.rs                 86bae8eaf2130c1e6ce48e5a5f9fdf5032ab542cfd02f636b51efa2c18f59688
      instructions.rs             2b5fd96a92f5c75fd5afac136220e2444454b24801def5f0360f242fcde04841
      mod.rs                      868504e67aec4a189c383214937557f42c048873043b686fc9d6cf49bbc5f898
```


B | Proof of Concepts

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

```
pocs/
  os-cvg-adv-00/
    hash             0427381fdc7c519b3eb647fed2ceca067c4c659721a66b272fd6d7f3f55c8f03
    patch            ea23bbfb9765f376cc04ef297580ae20f999a5f97bfffc3fca441fd48ff06e4b
    run.sh           52a74eed37d4cee96a21e3a50182b758785054e9b637c1d20409ec813d27eca9
```

C | 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 D.

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.

D | 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.

E | 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)