macro

Seven Seas A-29

Security Audit

Feb 21, 2025
Version 1.0.0

Presented by 0xMacro

Table of Contents

Introduction

This document includes the results of the security audit for Seven Seas's program code as found in the section titled ‘Source Code’. The security audit was performed by the Macro security team from Feb 10th to Feb 20th 2025.

The purpose of this audit is to review the source code of certain Seven Seas Solana programs, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

Disclaimer: While Macro’s review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.

Overall Assessment

The following is an aggregation of issues found by the Macro Audit team:

Severity	Count	Acknowledged	Won't Do	Addressed
High	1	-	-	1
Medium	3	-	-	3
Low	2	-	-	2
Code Quality	14	-	2	12

Seven Seas was quick to respond to these issues.

Specification

Our understanding of the specification was based on the following sources:

- Discussions with the Seven Seas team.
- Available documentation in the repository.

Source Code

The following source code was reviewed during the audit:

- **Repository:** boring-vault-svm
- **Commit Hash:** `ea1e9036856accfaaf2767835230547fb59530a0`

Specifically, we audited Solana programs that mimic Veda's Boring Vault contracts on the EVM chain.

Source Code	SHA256
boring-vault-svm/src/constants.rs	`df79f990986184edd3a89c3ab1050587269c7cbb23dca6dea9ae960d531dc0c3`
boring-vault-svm/src/error.rs	`3c989f44d563e38716904ee55c6d8095a77f6bd65d141b0efbbefa6a2191ef02`
boring-vault-svm/src/lib.rs	`beed56647a6440c9850c92fb84cc8563eea88b03188cac745fad44544c126b1d`
boring-vault-svm/src/state.rs	`9135996e1a4ab937d9ec33dcf70e2284accf4584682d03c8333418f3437e0922`
boring-vault-svm/src/utils/mod.rs	`8cae17fbd2a731c44189a9c5b249f15b135b6893db92e9db98198487316cdba8`
boring-vault-svm/src/utils/operators.rs	`8cdad473e1a1540ce63eee8eac2f45fb07d654956a241418be0a1d83bebc2281`
boring-vault-svm/src/utils/teller.rs	`5ce667fee3dcbbb26c2270e1baba2e896b85f528ec3aebbe15157cbd19493ada`

Source Code	SHA256
boring-onchain-queue/src/constants.rs	e93f16dcabada4003c3d8ef941cbfbd3c46d2368f5c885bf2d9751bce62101e9
boring-onchain-queue/src/error.rs	6547765064f2ebafa53fb6762835f56ca3a2a7fcd143fcf45d8081c4853f255f
boring-onchain-queue/src/lib.rs	404f726f872950a659391b7e81b420273eeb08f7ad4cb43c1cf6c4d1b1986e3a
boring-onchain-queue/src/state.rs	8e667d836eb1b5d5ce46f1685d942c8c79dd586ee5689ae9c8284bad5111a6ee
boring-onchain-queue/src/utils/mod.rs	3da9e41e54128210d823f125f1da814f694f96a0e8f982a53fd5500ae70bc06c
boring-onchain-queue/src/utils/utils.rs	b92ecfbad281624f171294ee2613aefbbdee47274bf85ec0112e102b816c63a2
boring-onchain-queue/src/utils/validate.rs	2c125bda79183630e5e1301da83a7dd99f97f694f59926423bb7eaa76cd84ef5

Note: This document contains an audit solely of the Solana programs listed above. Specifically, the audit pertains only to the programs themselves, and does not pertain to any other programs or scripts, including deployment scripts.

Issue Descriptions and Recommendations

Click on an issue to jump to it, or scroll down to see them all.

~~Q-14~~ Nitpicks

Security Level Reference

We quantify issues in three parts:

1. The high/medium/low/spec-breaking **impact** of the issue:
    - How bad things can get (for a vulnerability)
    - The significance of an improvement (for a code quality issue)
    - The amount of gas saved (for a gas optimization)
2. The high/medium/low **likelihood** of the issue:
    - How likely is the issue to occur (for a vulnerability)
3. The overall critical/high/medium/low **severity** of the issue.

This third part – the severity level – is a summary of how much consideration the client should give to fixing the issue. We assign severity according to the table of guidelines below:

Severity	Description
(C-x) Critical	We recommend the client **must** fix the issue, no matter what, because not fixing would mean **significant funds/assets WILL be lost.**
(H-x) High	We recommend the client **must** address the issue, no matter what, because not fixing would be very bad, *or* some funds/assets will be lost, *or* the code's behavior is against the provided spec.
(M-x) Medium	We recommend the client to **seriously consider** fixing the issue, as the implications of not fixing the issue are severe enough to impact the project significantly, albiet not in an existential manner.
(L-x) Low	The risk is small, unlikely, or may not relevant to the project in a meaningful way. Whether or not the project wants to develop a fix is up to the goals and needs of the project.
(Q-x) Code Quality	The issue identified does not pose any obvious risk, but fixing could improve overall code quality, on-chain composability, developer ergonomics, or even certain aspects of protocol design.
(I-x) Informational	Warnings and things to keep in mind when operating the protocol. No immediate action required.
(G-x) Gas Optimizations	The presented optimization suggestion would save an amount of gas significant enough, in our opinion, to be worth the development cost of implementing it.

Issue Details

~~H-1~~ Incorrect Decimal Scaling in Pegged Asset Deposits Leads to Inflated/Deflated Share Minting

TOPIC	STATUS	IMPACT	LIKELIHOOD
Decimals handling	Fixed	High ✳	Medium ✳

While depositing token to the vault, the `calculate_shares_and_mint()` function contains a critical decimal handling error when processing deposits of pegged assets with different decimal places than the base asset.

```
pub fn calculate_shares_and_mint<'a>(
    is_base: bool,
    args: DepositArgs,
    exchange_rate: u64,
    share_decimals: u8,
    asset_decimals: u8,
    asset_data: Account<'_, AssetData>,
    price_feed: AccountInfo<'a>,
    token_2022: AccountInfo<'a>,
    share_mint: AccountInfo<'a>,
    user_shares: AccountInfo<'a>,
    boring_vault_state: AccountInfo<'a>,
    boring_vault_state_bump: u8,
) -> Result<u64> {

    ...

    } else if asset_data.is_pegged_to_base_asset {
        // Asset is pegged to base asset, so just need to convert amount t
        let deposit_amount = to_decimal(args.deposit_amount, asset_decima
        let deposit_amount: u64 = from_decimal(deposit_amount, share_deci

        calculate_shares_to_mint_using_base_asset(
            deposit_amount,
            exchange_rate,
            asset_decimals,
```

```
                share_decimals,
                asset_data.share_premium_bps,
            )?
        }

        ...
    }
```

Reference: boring-vault-svm/src/utils/teller.rs#L182-L193

The function first converts the deposit amount to the base asset's decimal places but then incorrectly uses the deposit asset's decimals when calculating shares in `calculate_shares_to_mint_using_base_asset`. This causes a severe multiplication effect when the base asset has different decimal places than the deposit asset, leading to the depositor having far more or far fewer share tokens than expected

For example, when depositing `USDC` (6 decimals) to a vault using `USDT` (9 decimals) (hypothetically, as USDT is not 9 decimals) as the base token:

1. User deposits 1 USDC (1,000,000 units)
2. The amount is converted to USDT decimals: 1,000,000,000 units
3. When calculating shares, this amount is interpreted as if it were in USDC decimals (6)
4. Results in 1000x more shares being minted than intended

This allows users to receive significantly more shares than they should, effectively stealing value from other vault depositors.

- POC

  Put this test in a new folder and run `anchor test`

```
 import * as anchor from "@coral-xyz/anchor";
import { Program } from "@coral-xyz/anchor";
import {
  createMint,
  getAccount,
  getOrCreateAssociatedTokenAccount,
```

```
  mintTo,
  TOKEN_PROGRAM_ID,
  TOKEN_2022_PROGRAM_ID,
  ASSOCIATED_TOKEN_PROGRAM_ID,
  getAssociatedTokenAddressSync,
} from "@solana/spl-token";

import { BoringVaultSvm } from "../target/types/boring_vault_svm";
import { expect } from "chai";

describe("Test", () => {
  // anchor
  let provider;
  let program;
  let user;

  const LAMPORTS_PER_SOL = 1000000000;

  let programConfigAccount;
  let boringVaultStateAccount;
  let boringVaultShareMint;
  let boringVaultAccount;
  let userShareAta;

  let mintUSDC;
  let mintUSDT;

  let userUSDCATA;
  let userUSDTATA;

  let usdcAssetDataPda;
  let usdtAssetDataPda;

  let vaultUSDCATA;
  let vaultUSDTATA;

  let _;

  before(async () => {
    anchor.setProvider(anchor.AnchorProvider.env());
    provider = anchor.getProvider();
    program = anchor.workspace.BoringVaultSvm as Program<BoringVaultS
    user = anchor.web3.Keypair.generate();

    await provider.connection.confirmTransaction(
      await provider.connection.requestAirdrop(
        user.publicKey,
        100 * LAMPORTS_PER_SOL
      )
    );

    mintUSDC = await createMint(
```

```
    provider.connection,
    user,
    user.publicKey,
    user.publicKey,
    6
  );

  mintUSDT = await createMint(
    provider.connection,
    user,
    user.publicKey,
    user.publicKey,
    9
  );

  [programConfigAccount, _] = anchor.web3.PublicKey.findProgramAddre
    [Buffer.from("config")],
    program.programId
  );
  [boringVaultStateAccount, _] = anchor.web3.PublicKey.findProgramA
    [Buffer.from("boring-vault-state"), Buffer.from(new Array(8).fi
    program.programId
  );

  [boringVaultShareMint, _] = anchor.web3.PublicKey.findProgramAddre
    [Buffer.from("share-token"), boringVaultStateAccount.toBuffer()
    program.programId
  );

  [usdcAssetDataPda, _] = anchor.web3.PublicKey.findProgramAddressSy
    [
      Buffer.from("asset-data"),
      boringVaultStateAccount.toBuffer(),
      mintUSDC.toBuffer(),
    ],
    program.programId
  );
  [usdtAssetDataPda, _] = anchor.web3.PublicKey.findProgramAddressSy
    [
      Buffer.from("asset-data"),
      boringVaultStateAccount.toBuffer(),
      mintUSDT.toBuffer(),
    ],
    program.programId
  );
  [boringVaultAccount, _] = anchor.web3.PublicKey.findProgramAddress
    [
      Buffer.from("boring-vault"),
      Buffer.from(new Array(8).fill(0)),
      Buffer.from([0]),
    ],
    program.programId
```

```
);

userUSDCATA = await getOrCreateAssociatedTokenAccount(
  provider.connection,
  user,
  mintUSDC,
  user.publicKey
);

userUSDTATA = await getOrCreateAssociatedTokenAccount(
  provider.connection,
  user,
  mintUSDT,
  user.publicKey
);

vaultUSDCATA = await getOrCreateAssociatedTokenAccount(
  provider.connection,
  user,
  mintUSDC,
  boringVaultAccount,
  true
);
vaultUSDTATA = await getOrCreateAssociatedTokenAccount(
  provider.connection,
  user,
  mintUSDT,
  boringVaultAccount,
  true
);

//mint 1$ worth of USDC
await mintTo(
  provider.connection,
  user,
  mintUSDC,
  userUSDCATA.address,
  user,
  1_000_000
);

//mint 1$ worth of USDT
await mintTo(
  provider.connection,
  user,
  mintUSDT,
  userUSDTATA.address,
  user,
  1_000_000_000
);

await program.methods
```

```
  .initialize(user.publicKey)
  .accounts({
    config: programConfigAccount,
    signer: user.publicKey,
    systemProgram: anchor.web3.SystemProgram.programId,
  })
  .signers([user])
  .rpc();

await provider.connection.confirmTransaction(
  await program.methods
    .deploy({
      authority: user.publicKey,
      name: "Boring Vault",
      symbol: "BV",
      exchangeRateProvider: user.publicKey,
      exchangeRate: new anchor.BN(1_000_000_000), // 9 decimals ca
      payoutAddress: user.publicKey,
      allowedExchangeRateChangeUpperBound: 10050,
      allowedExchangeRateChangeLowerBound: 9950,
      minimumUpdateDelayInSeconds: 0,
      platformFeeBps: 0,
      performanceFeeBps: 0,
      strategist: user.publicKey,
      withdrawAuthority: anchor.web3.PublicKey.default, // permiss
    })
    .accounts({
      config: programConfigAccount,
      boringVaultState: boringVaultStateAccount,
      shareMint: boringVaultShareMint,
      baseAsset: mintUSDT,
      signer: user.publicKey,
      systemProgram: anchor.web3.SystemProgram.programId,
      tokenProgram: TOKEN_2022_PROGRAM_ID,
    })
    .signers([user])
    .rpc()
);

userShareAta = getAssociatedTokenAddressSync(
  boringVaultShareMint,
  user.publicKey,
  true,
  TOKEN_2022_PROGRAM_ID
);

await program.methods
  .updateAssetData({
    vaultId: new anchor.BN(0),
    assetData: {
      allowDeposits: true,
      allowWithdrawals: true,
```

```
          sharePremiumBps: 0,
          isPeggedToBaseAsset: true,
          priceFeed: anchor.web3.PublicKey.default,
          inversePriceFeed: false,
        },
      })
      .accounts({
        signer: user.publicKey,
        boringVaultState: boringVaultStateAccount,
        systemProgram: anchor.web3.SystemProgram.programId,
        asset: mintUSDC,
        assetData: usdcAssetDataPda,
      })
      .signers([user])
      .rpc();

    await program.methods
      .updateAssetData({
        vaultId: new anchor.BN(0),
        assetData: {
          allowDeposits: true,
          allowWithdrawals: true,
          sharePremiumBps: 0,
          isPeggedToBaseAsset: true,
          priceFeed: anchor.web3.PublicKey.default,
          inversePriceFeed: false,
        },
      })
      .accounts({
        signer: user.publicKey,
        boringVaultState: boringVaultStateAccount,
        // @ts-ignore
        systemProgram: anchor.web3.SystemProgram.programId,
        asset: mintUSDT,
        assetData: usdtAssetDataPda,
      })
      .signers([user])
      .rpc();
  });

  it("Depositing 1$ USDC get 1e12 share", async () => {
    // deposting USDC with 6 decimals
    await program.methods
      .deposit({
        vaultId: new anchor.BN(0),
        depositAmount: new anchor.BN(1_000_000),
        minMintAmount: new anchor.BN(0),
      })
      .accounts({
        // @ts-ignore
        signer: user.publicKey,
        boringVaultState: boringVaultStateAccount,
```

```
        boringVault: boringVaultAccount,
        depositMint: mintUSDC,
        assetData: usdcAssetDataPda,
        userAta: userUSDCATA.address,
        vaultAta: vaultUSDCATA.address,
        tokenProgram: TOKEN_PROGRAM_ID,
        tokenProgram2022: TOKEN_2022_PROGRAM_ID,
        systemProgram: anchor.web3.SystemProgram.programId,
        associatedTokenProgram: ASSOCIATED_TOKEN_PROGRAM_ID,
        shareMint: boringVaultShareMint,
        userShares: userShareAta,
        priceFeed: anchor.web3.PublicKey.default,
      })
      .signers([user])
      .rpc();

    let shareAccountData = await getAccount(
      provider.connection,
      userShareAta,
      undefined,
      TOKEN_2022_PROGRAM_ID
    );
    let shareBalance = shareAccountData.amount;
    expect(shareBalance).to.equal(BigInt(1_000_000_000_000)); //-> sh
  });

  it("Depositing 1$ USDT get 1e9 share", async () => {
    // deposting USDT with 9 decimals
    await program.methods
      .deposit({
        vaultId: new anchor.BN(0),
        depositAmount: new anchor.BN(1_000_000_000),
        minMintAmount: new anchor.BN(0),
      })
      .accounts({
        signer: user.publicKey,
        boringVaultState: boringVaultStateAccount,
        boringVault: boringVaultAccount,
        depositMint: mintUSDT,
        assetData: usdtAssetDataPda,
        userAta: userUSDTATA.address,
        vaultAta: vaultUSDTATA.address,
        tokenProgram: TOKEN_PROGRAM_ID,
        tokenProgram2022: TOKEN_2022_PROGRAM_ID,
        systemProgram: anchor.web3.SystemProgram.programId,
        associatedTokenProgram: ASSOCIATED_TOKEN_PROGRAM_ID,
        shareMint: boringVaultShareMint,
        userShares: userShareAta,
        priceFeed: anchor.web3.PublicKey.default,
      })
      .signers([user])
      .rpc();
```

```
        let shareAccountData = await getAccount(
          provider.connection,
          userShareAta,
          undefined,
          TOKEN_2022_PROGRAM_ID
        );
        let shareBalance = shareAccountData.amount;
        expect(shareBalance).to.equal(BigInt(1_001_000_000_000)); //-> 1e
      });
    });
```

Remediations to Consider

Consider fixing the issue in one of these two ways:

```
} else if asset_data.is_pegged_to_base_asset {
    let deposit_amount = to_decimal(args.deposit_amount, asset_decimals)?
    let deposit_amount: u64 = from_decimal(deposit_amount, share_decimals

    calculate_shares_to_mint_using_base_asset(
        deposit_amount,
        exchange_rate,
-       asset_decimals,
+       share_decimals,
        share_decimals,
        asset_data.share_premium_bps,
    )?
}
```

or

```
} else if asset_data.is_pegged_to_base_asset {
-   let deposit_amount = to_decimal(args.deposit_amount, asset_decimals)?
-   let deposit_amount: u64 = from_decimal(deposit_amount, share_decimals

    calculate_shares_to_mint_using_base_asset(
-       deposit_amount,
+       args.deposit_amount,
        exchange_rate,
        asset_decimals,
        share_decimals,
        asset_data.share_premium_bps,
    )?
}
```

~~M-1~~ Insufficient Account Space Allocation for CPI Digest Operations Vector

TOPIC	STATUS	IMPACT	LIKELIHOOD
Account Storage	Fixed	Medium ✳	Medium ✳

In the boring vault program, the `updraw_cpi_digest()` function initializes an account of type `CpiDigest` :

```
pub struct UpdateCpiDigest<'info> {
    ...

    #[account(
        init_if_needed,
        payer = signer,
        space = 8 + std::mem::size_of::<CpiDigest>(),
        seeds = [
            BASE_SEED_CPI_DIGEST,
            &args.vault_id.to_le_bytes()[..],
            args.cpi_digest.as_ref(),
        ],
        bump,
    )]
    pub cpi_digest: Account<'info, CpiDigest>,
}
```

Reference: boring-vault-svm/src/lib.rs#L1530-L1555

The `CpiDigest` struct contains a vector of operations:

```
#[account]
#[derive(Debug)]
pub struct CpiDigest {
    pub operators: Operators,
    pub expected_size: u16,
}

#[derive(AnchorSerialize, AnchorDeserialize, Clone, Debug)]
pub struct Operators {
    pub operators: Vec<Operator>,
```

```
}

#[derive(AnchorSerialize, AnchorDeserialize, Clone, Debug)]
pub enum Operator {
    Noop,
    IngestInstruction(u32, u8), // (ix_index, length)
    IngestAccount(u8),          // (account_index)
}
```

Reference: boring-vault-svm/src/state.rs#L131-L136 , boring-vault-svm/src/utils/operators.rs#L4-L14

While using `std::mem::size_of<CpiDigest>()` for space allocation is generally good practice, it becomes problematic when the struct contains a vector field. In Solana, vectors are stored as contiguous byte arrays with a 4-byte length prefix, unlike standard Rust, where vectors are pointer-based structures.

Let's analyze the available space:

- `std::mem::size_of<Operator>()` = 24 bytes
- 4 bytes for vector length prefix
- Leaving only 20 bytes for actual vector data

However, each operation in the vector requires:

- 1 byte for enum variant discriminator
- Up to 5 bytes for data (`IngestInstruction` stores u32 + u8)

With only 20 bytes available for vector storage, the account can store at most 3 `IngestInstruction` operations. This is likely insufficient for complex vault management operations, causing account creation to fail when attempting to store larger operation sets.

Remediations to Consider

Since the program includes a `close_cpi_digest()` function to clean up accounts, it's acceptable to allocate more space than strictly needed. Consider making the account size configurable

~~M-2~~ Missing Pegged Asset Case in Exchange Rate Calculation in Boring Queue

TOPIC	STATUS	IMPACT	LIKELIHOOD
Missing logic	Fixed	Medium ✳	High ✳

When requesting a withdrawal to quote token in the Boring Queue program, the `request_withdraw()` function makes a CPI (Cross Program Invocation) to the Boring Vault's `get_rate_in_quote_safe()` function to fetch the current exchange rate of quote token and share token. However, it fails to handle cases where the quote token is pegged to the base asset.

```
pub fn get_rate_in_quote(
    boring_vault_state: Account<'_, BoringVault>,
    quote: InterfaceAccount<'_, Mint>,
    asset_data: Account<'_, AssetData>,
    price_feed: AccountInfo,
) -> Result<u64> {
    if boring_vault_state.teller.base_asset == quote.key() {
        get_rate(boring_vault_state)
    } else {
            //**MISSING THE PEGGED TOKEN HANDLING HERE**

        let feed_account = price_feed.data.borrow();
        let feed = PullFeedAccountData::parse(feed_account).unwrap();

        let price = match feed.value() {
            Some(value) => value,
            None => return Err(BoringErrorCode::InvalidPriceFeed.into()),
        };

        let price = if asset_data.inverse_price_feed {
            Decimal::from(1).checked_div(price).unwrap()
        } else {
            price
        };

        let exchange_rate = to_decimal(
            boring_vault_state.teller.exchange_rate,
            boring_vault_state.teller.decimals,
        )?;
```

```
            let rate = exchange_rate.checked_div(price).unwrap();

            // Scale rate to quote decimals.
            let rate = from_decimal(rate, quote.decimals)?;

            Ok(rate)
        }
    }
```

Reference: boring-vault-svm/src/utils/teller.rs#L299-L338

The function currently only handles two scenarios:

- Quote token is the base asset - uses a direct exchange rate
- Quote token is not the base asset - attempts to fetch external price data

This creates a problem because pegged tokens don't have a direct price feed between them, yet they need a special conversion path that accounts for their pegged relationship and different decimals. When attempting to withdraw using a pegged token, the function will try to fetch a price feed that doesn't exist (since the price feed account will likely be a default address). This will cause the withdrawal to fail.

This effectively makes it a DOS to request withdrawal using pegged tokens. The vault admin can hotfix it by changing the price feed by calling to `update_asset_data()` , but it can depend on the availability of the price feed at that current time

Remediations to Consider

Consider adding a logic path for the pegged token in the `get_rate_in_quote()` function

---

~~M-3~~ Missing Price Feed Staleness Check Allows Trading with Outdated Oracle Data

TOPIC	STATUS	IMPACT	LIKELIHOOD
Oracle staleness	Fixed	High	Low

The vault's price feed implementation in the Boring Vault program lacks staleness checks when fetching oracle prices for non-pegged assets. This could allow trades to execute with outdated price data in scenarios where the Switchboard Oracle feed becomes stale, potentially leading to incorrect share calculations and economic losses.

The vulnerability exists in both deposit and withdrawal flows where price feeds are used:

- In `calculate_shares_and_mint()` function for deposits of non-pegged assets
- In `calculate_assets_out()` function for withdrawals of non-pegged assets

Remediations to Consider

Consider following this example provided by the Switchboard team:

```
use switchboard_on_demand::PullFeedAccountData;
use rust_decimal::Decimal;

pub fn solana_ix<'a>(mut ctx: Context< YourAccounts<'a>>, params: Params)
    // Assume `account_info` is obtained from the Solana blockchain
    let feed = PullFeedAccountData::parse(ctx.accounts.sb_feed)?;
    let max_stale_slots = 100; // Define the maximum number of slots befo
    let min_samples = 5; // Set the minimum number of samples for data ac
    let price: Decimal = feed.get_value(&Clock::get()?, max_stale_slots,

    msg!("Oracle Price: {}", price);

    Ok(())
}
```

~~L-1~~ Unnecessary pause check when closing CPI digest accounts

TOPIC	STATUS	IMPACT	LIKELIHOOD
Over constrained	Fixed	Low	Low

The `CloseCpiDigest` struct includes a pause check that prevents closing CPI digest accounts when the vault is paused:

```
#[account(
    seeds = [BASE_SEED_BORING_VAULT_STATE, &args.vault_id.to_le_bytes()[.
    bump,
    constraint = boring_vault_state.config.paused == false @ BoringErrorC
    constraint = signer.key() == boring_vault_state.config.authority.key(
)]
pub boring_vault_state: Account<'info, BoringVault>,
```

Reference: boring-vault-svm/src/lib.rs#L1563-L1569

This is overly restrictive since:

- Closing accounts is an administrative cleanup action
- The authority check already provides security
- Closing CPI digests doesn't affect vault assets

Remediations to Consider

Consider removing the pause check constraint and keeping only the authority verification.

~~L-2~~ Missing maximum deadline validation allows users to lock their funds indefinitely

TOPIC	STATUS	IMPACT	LIKELIHOOD
Sanity check	Fixed	Medium	Low

The `request_withdraw()` function only validates minimum deadline through `minimum_seconds_to_deadline` but lacks maximum deadline validation. Users can set arbitrarily long deadlines when requesting withdrawals, potentially locking their share tokens in the queue contract for extended periods if the request cannot be fulfilled by a solver (e.g. due to paused queue, etc).

Remediations to Consider

Consider adding a `maximum_seconds_to_deadline` parameter to `WithdrawAssetData` and validate against it in `request_withdraw()`

~~Q-1~~ Inconsistent validation of strategist address between `deploy()` and `set_strategist()`

TOPIC	STATUS	QUALITY IMPACT
Sanity check	Fixed	Low

In `set_strategist()`, there is a validation check to prevent setting the strategist address to the default address. However, this same validation is missing in the `deploy()` function when initially setting the strategist. Consider adding the same validation check in the `deploy()`

~~Q-2~~ Inconsistent validation of exchange rate provider address

TOPIC	STATUS	QUALITY IMPACT
Sanity check	Fixed	Low

In `deploy()`, the exchange rate provider address is validated to not be the default address, but this check is missing in `update_exchange_rate_provider()`.

This creates inconsistency in validation logic across the codebase. Consider adding the same validation check in the `update_exchange_rate_provider()` function

---

~~Q-3~~ Unnecessary Clock Sysvar Account Inclusion in Instructions

TOPIC	STATUS	QUALITY IMPACT
Best practice	Fixed	Low

Multiple instructions in the program unnecessarily include the Clock Sysvar account in their instruction contexts when they could directly access it within the function using `Clock::get()?` . This includes Boring Queue's `RequestWithdraw`, `CancelWithdraw`, `FulfillWithdraw`, and Boring Vault's `Deploy`. Consider removing the Clock Sysvar account from the instructions and replacing it with direct access

---

~~Q-4~~ Missing sanity check on `update_withdraw_asset_data()`

TOPIC	STATUS	QUALITY IMPACT
Sanity check	Fixed	Low

In `update_withdraw_asset_data()` , there is no validation to ensure that `maximum_discount` is greater than `minimum_discount` . This could lead to a situation where the withdraw queue becomes unusable if misconfigured, as no valid discount value would exist that satisfies both minimum and maximum constraints. Consider adding the validation check.

~~Q-5~~ Allow Default Price Feed for Base Asset

TOPIC	STATUS	QUALITY IMPACT
Sanity check	Fixed	Low

In the `update_asset_data()` function, the price feed can be set to the default value only if `is_pegged_to_base_asset` is true. However, the base asset itself should also be allowed to have a default price feed since it doesn't need price conversion. Consider adding a check to allow default price feed for the base asset:

```
    if args.asset_data.price_feed == Pubkey::default() {
        require!(
            args.asset_data.is_pegged_to_base_asset ||
+               asset.key() == boring_vault_state.teller.base_asset,
            BoringErrorCode::InvalidPriceFeed
        );
    }
```

Reference: boring-vault-svm/src/lib.rs#L243-L248

~~Q-6~~ Inconsistent PDA Seed Structure Across Programs

TOPIC	STATUS	QUALITY IMPACT
Best practice	Fixed	Low

The programs use inconsistent seed ordering and derivation patterns for PDAs (Program Derived Addresses):

- In the Boring Vault program, `BoringVaultState`, `CPIDigest`, and `BoringVault` accounts use the vault's ID to derive to those accounts, while `AssetData` and

share mint accounts use the vault's address to derive

- In the Boring Queue program, `WithdrawAssetData` accounts use the vault's ID as the third seed, while other accounts use the vault's ID as the second seed

Consider making it more consistent for better maintainability and to reduce the potential for integration errors.

~~Q-7~~ Unnecessary `#[account(mut)]` field-level attributes

TOPIC	STATUS	QUALITY IMPACT
Best practice	Fixed	Low

Throughout the programs, some of the accounts in instructions unnecessarily use `#[account(mut)]` attributes. Here's the exhaustive list:

- In the Boring Vault program:
  - `signer` account in `Pause`, `Unpause`, `TransferAuthority`, `AcceptAuthority`, `UpdateExchangeRateProvider`, `SetWithdrawSubAccount`, `SetPayout`, `SetFees`, `SetStrategist`, `ClaimFeesInBase`, `UpdateExchangeRate` instructions
  - `boring_vault` account in `ClaimFeesInBase`, `Deposit`, `Withdraw` instructions
  - `boring_vault_state` account in `Deposit`, `Withdraw` instructions
- In the Boring Queue program:
  - `signer` account in `SetSolveAuthority`, `Pause`, `Unpause` instructions
  - `queue` account in `RequestWithdraw`, `CancelWithdraw` instructions
  - `share_mint` account in `RequestWithdraw`, `CancelWithdraw` instructions

- `boring_vault_state` , `boring_vault` accounts in `FulfillWithdraw` instruction

Consider removing those attributes

~~Q-8~~ No need `init_if_needed` to user's share account in `withdraw()`

TOPIC	STATUS	QUALITY IMPACT
Best practice	Fixed	Low

When withdrawing from the boring vault, the user's share account will be used to burn the share token. This means that the share account is already initialized and no need for the `init_if_needed` tag. The initialization logic can be safely removed and replaced with the `mut` attribute:

```
    #[account(
-       init_if_needed,
-       payer = signer,
+       mut,
        associated_token::mint = share_mint,
        associated_token::authority = signer,
        associated_token::token_program = token_program_2022,
    )]
    pub user_shares: InterfaceAccount<'info, TokenAccount>,
```

Reference: boring-vault-svm/src/lib.rs#L1511-L1518

~~Q-9~~ Missing validation for `share_premium_bps` upper bound in `update_asset_data()` function

TOPIC	STATUS	QUALITY IMPACT
Sanity check	Fixed	Low

The `update_asset_data()` function allows setting `share_premium_bps` without any upper bound validation, while the EVM implementation enforces a 1000 bps (10%) limit. This inconsistency between implementations could lead to confusion for protocol integrators. Consider adding a similar upper-bound check

---

~~Q-10~~ Multiple accounts in instructions not used

TOPIC	STATUS	QUALITY IMPACT
Best practice	Fixed	Low

Several instructions in the program include accounts that are never utilized in their execution. This creates unnecessary overhead in transaction size and validation. Here's the exhaustive list:

- In the Boring Vault program:
  - `token_program` account in `Deploy` instruction
  - `associated_token_program` account in `ClaimFeesInBase` , `DepositSol` , `Deposit` , `Withdraw` instructions
  - `system_program` account in `ViewCpiDigest` instruction
- In the Boring Queue program:
  - `queue` account in `Deploy` instruction
  - `associated_token_program` account in `CancelWithdraw` , `FulfillWithdraw` instructions

Consider removing unused account declarations from the affected instructions to improve code clarity and reduce transaction overhead

---

Q-11 No need to set some default state

TOPIC	STATUS	QUALITY IMPACT
Best practices	Wont Do	Low

Multiple instances across the codebase where variables are explicitly initialized to their default values, which is redundant since Rust automatically initializes variables to their type's default value. Here's the exhaustive list:

- boring-vault-svm/src/lib.rs#L52
- boring-vault-svm/src/lib.rs#L94
- boring-vault-svm/src/lib.rs#L105
- boring-vault-svm/src/lib.rs#L106 (set to the share token's current total supply, which is 0)
- boring-onchain-queue/src/lib.rs#L87
- boring-onchain-queue/src/lib.rs#L165

RESPONSE BY SEVEN SEAS

> Won't do, the default state setting will be left in, to make it very explicit what state we want used for the default.

Q-12 Discount range mechanism is not relevant for Solana

implementation

TOPIC	STATUS	QUALITY IMPACT
Redundant logic	Wont Do	Low

The program implements a discount range mechanism where withdrawers must specify a discount between `minimum_discount` and `maximum_discount` when requesting withdrawals. This design was carried over from EVM implementation where discounts incentivize solvers by covering their gas costs.

However, in the Solana implementation, the discount amount is sent to the vault instead of the solvers. Solvers are compensated through rent from closing the withdrawal request account rather than through discounts. This makes the flexible discount range unnecessary since it no longer serves its original purpose.

Consider simplifying the design by using a fixed discount rate instead of a range, since the flexibility adds complexity without providing benefit in the Solana context.

RESPONSE BY SEVEN SEAS

> Won't do, it's true this isn't really needed but in an effort to have the code more closely resemble the EVM code I think it is fine to leave this.

~~Q-13~~ Inconsistent Error Handling Patterns

TOPIC	STATUS	QUALITY IMPACT
Best practice	Fixed	Low

The codebase uses two different patterns for handling validation errors:

1. The `require!` macro pattern, for example:

```
require!(
    expected_share_mint == args.share_mint,
    QueueErrorCode::InvalidShareMint
);
```

2. The if-statement with an error return pattern, for example:

```
if current_time < maturity {
    return Err(QueueErrorCode::RequestNotMature.into());
}
```

Consider using only one pattern for a better uniform codebase

---

~~Q-14~~ Nitpicks

TOPIC	STATUS	QUALITY IMPACT
Nitpicks	Fixed	Low

1. The `name` and `symbol` fields in the `DeployArgs` struct are never used in the program, creating unnecessary bloat. Consider removing them
2. `close_cpi_digest()` reuses `UpdateCpiDigestArgs` struct but doesn't utilize its `operators` and `expected_size` fields when a simpler dedicated struct would suffice.
3. `Pause` and `Unpause` structs have identical implementations and could be consolidated into a single struct.
4. The `#[instruction(args: DeployArgs)]` annotation on the `Deploy` struct is unnecessary since the arguments are not used in any constraints. It can be safely removed.
5. Withdraw mint validation in the `fulfill_withdraw()` function should be moved to account constraints for better efficiency and clarity in the

validation flow.

6. `require_keys_neq!` check in `boring-vault-svm` lacks a custom error code, reducing error handling specificity.

7. Empty `#[account()]` attribute in the `cpi_digest` account adds no value and can be safely removed.

8. `pending_authority` parameter in the `TransferAuthority` struct is unused and should be removed to improve code cleanliness.

Disclaimer

Macro makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and Macro specifically disclaims all implied warranties of merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

Macro will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand by any other party. In no event will Macro be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if Macro has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the Seven Seas team and only the source code Macro notes as being within the scope of Macro's review within this report. This report does not include an audit of the deployment scripts used to deploy the Solana programs in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than Macro. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that Macro is not responsible for the content or operation of such websites, and that Macro shall have no liability to your or any other person or entity for the use of third party websites. Macro assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.