SHERLOCK

Security Review For ApeBond

Collaborative Audit Prepared For: **ApeBond**
Lead Security Expert(s): **0xeix**
Date Audited: **May 26 - June 9, 2025**
Final Commit: **95ff38e**

Introduction

Apebond is a Solana-based program built with Anchor that facilitates the issuance and management of on-chain bonds. It is an adaptation of the Apebond system for EVM, now implemented on Solana. This security review is a focused and extensive dive into ApeBond's Solana Bonds programs.

Scope

Repository: ApeSwapFinance/apebond-solana

Audited Commit: defda35471cc7c87f60d9ed68f2cb66f88e20ac1

Final Commit: 95ff38ed42c622cd37964354d7f757b07d1d5ea2

Files:

- programs/apebond/src/constants.rs
- programs/apebond/src/errors.rs
- programs/apebond/src/instructions/claim.rs
- programs/apebond/src/instructions/claimable_payout.rs
- programs/apebond/src/instructions/close_treasury.rs
- programs/apebond/src/instructions/current_debt.rs
- programs/apebond/src/instructions/debt_decay.rs
- programs/apebond/src/instructions/debt_ratio.rs
- programs/apebond/src/instructions/deposit.rs
- programs/apebond/src/instructions/initialize_bond_issuance.rs
- programs/apebond/src/instructions/initialize_bond_pricing.rs
- programs/apebond/src/instructions/initialize_bond_term.rs
- programs/apebond/src/instructions/initialize_collection.rs
- programs/apebond/src/instructions/issuance_control.rs
- programs/apebond/src/instructions/mod.rs
- programs/apebond/src/instructions/payout_for.rs
- programs/apebond/src/instructions/payouts_current.rs
- programs/apebond/src/instructions/pending_payout.rs
- programs/apebond/src/instructions/pending_vesting.rs
- programs/apebond/src/instructions/permissioned_account.rs
- programs/apebond/src/instructions/set_bcv.rs

- programs/apebond/src/instructions/set_bcv_update_interval.rs
- programs/apebond/src/instructions/set_bond_status.rs
- programs/apebond/src/instructions/set_bond_terms.rs
- programs/apebond/src/instructions/set_claim_approval.rs
- programs/apebond/src/instructions/set_fee.rs
- programs/apebond/src/instructions/set_fee_recipients.rs
- programs/apebond/src/instructions/set_max_total_payout.rs
- programs/apebond/src/instructions/set_min_price.rs
- programs/apebond/src/instructions/transfer_payout_token.rs
- programs/apebond/src/instructions/true_bond_price.rs
- programs/apebond/src/instructions/value_of_token.rs
- programs/apebond/src/instructions/vested_payout_at_time.rs
- programs/apebond/src/instructions/vesting_period.rs
- programs/apebond/src/lib.rs
- programs/apebond/src/processing/claim_calculations.rs
- programs/apebond/src/processing/mod.rs
- programs/apebond/src/processing/price_calculations.rs
- programs/apebond/src/processing/time_calculations.rs
- programs/apebond/src/state/bond.rs
- programs/apebond/src/state/bond_issuance.rs
- programs/apebond/src/state/bond_pricing.rs
- programs/apebond/src/state/bond_term.rs
- programs/apebond/src/state/claim_approval.rs
- programs/apebond/src/state/issuance_control.rs
- programs/apebond/src/state/mod.rs
- programs/apebond/src/state/permissioned_account.rs

Final Commit Hash

95ff38ed42c622cd37964354d7f757b07d1d5ea2

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
3	3	4

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: Price is not computed based on the decayed total debt

Source: https://github.com/sherlock-audit/2025-05-apebond-may-26th/issues/49

Vulnerability Detail

In the Solidity version of the ApeBond protocol, when the `_decayDebt()` is executed, the global `totalDebt` variable is reduced immediately. For instance, if the initial `totalDebt` is 100, after the `_decayDebt()` is executed, it is reduced to 75. Subsequently, the `trueBillPrice()` is executed, which internally relies on the `currentDebt()` to compute the price. So, the price is calculated based on the reduced `totalDebt` (75) over here.

```
File: ApeBond.sol
    function currentDebt() public view returns (uint256) {
        return totalDebt - debtDecay();
    }
```

However, in the Solana implementation, in Line 322 below, the code computes the new/reduced total debt and stores the reduced value (75) in a temporary variable `new_total_debt`. Note that this value has not been written into the storage yet (`self.bond_pricing.total_debt`). It will only be written to the storage towards the end of the function when `ctx.accounts.update_accounts()` is executed. Thus, in Line 330 below, the `self.bond_pricing.total_debt` remains at 100 (not reduced).

https://github.com/sherlock-audit/2025-05-apebond-may-26th/blob/633b78dc37b158832f43896f7253166013df40b5/apebond-solana/programs/apebond/src/instructions/deposit.rs#L305

```
File: deposit.rs
305: pub fn deposit(
306:     ctx: Context<Deposit>,
307:     amount: u64,
308:     max_price: u64,
309: ) -> Result<()> {
..SNIP..
320:     // _decayDebt() : Calculate new total debt
321:     // totalDebt += _amount;   // Increase totalDebt by amount deposited
322:     let new_total_debt = ctx.accounts.bond_pricing.total_debt
323:     .checked_add(amount)
324:     .ok_or(ApeBondError::ArithmeticOverflow)?
325:     .checked_sub(debt_decay)
326:     .ok_or(ApeBondError::ArithmeticOverflow)?;
327:     ctx.accounts.bond_pricing.last_decay = current_timestamp;
328:
329:     // Calculate true price first
330:     let true_price = true_bond_price(
```

```
331:         &ctx.accounts.bond_pricing,
332:         &ctx.accounts.bond_term,
333:         current_timestamp,
334:         ctx.accounts.bond_issuance.principal_mint_decimals,
335:         ctx.accounts.bond_issuance.payout_mint_decimals,
336:     )?;
```

Then, when computing the `true_bond_price()`, the `debt_ratio()` -> `current_debt()` will be called. Another problem is that in Line 327 above, it sets the `ctx.accounts.bond_pricing.last_decay` to the current timestamp. Thus, when computing the `debt_decay()` below in Line 54, the `last_decay == current_timestamp`, so there is no decay (`decay_amount` = 0)

So, in Line 62 below, `total_debt (100) - decay_amount (0) = 100`. Thus, the price is computed using the original total debt of 100.

https://github.com/sherlock-audit/2025-05-apebond-may-26th/blob/633b78dc37b158832f43896f7253166013df40b5/apebond-solana/programs/apebond/src/processing/price_calculations.rs#L62

```
File: price_calculations.rs
41: /// Calculates the current debt by subtracting decay amount from total debt
42: pub fn current_debt(
43:     total_debt: u64,
44:     last_decay: u64,
45:     current_timestamp: u64,
46:     vesting_end: u64,
47: ) -> Result<u64> {
48:     // If vesting_end is 0, return total debt
49:     if vesting_end == 0 {
50:         return Ok(total_debt);
51:     }
52:
53:     // Calculate decay amount
54:     let decay_amount = debt_decay(
55:         total_debt,
56:         last_decay,
57:         current_timestamp,
58:         vesting_end
59:     )?;
60:
61:     // Calculate current debt by subtracting decay amount
62:     Ok(total_debt.checked_sub(decay_amount).ok_or::<Error>(ApeBondError::ArithmeticOverflow.into())?)
63: }
```

Solidity computes price using the updated total debt of 75, while Solana computes price using the old/original debt of 100. The correct approach is to compute the price based on the total debt after decay, similar to what has been done in the Solidity implementation.

Impact

The computed price will be inaccurate, resulting in the incorrect number of payout tokens being issued to users during or after the vesting period.

Code Snippet

https://github.com/sherlock-audit/2025-05-apebond-may-26th/blob/633b78dc37b158832f43896f7253166013df40b5/apebond-solana/programs/apebond/src/instructions/deposit.rs#L305

Recommendation

The `new_total_debt` value should be updated immediately into the `self.bond_pricing.total_debt` (right away after setting the decay timestamp) before computing the price via the `true_bond_price()` function.

Issue H-2: Treasury cannot be closed if there is a remaining SPL token balance

Source: https://github.com/sherlock-audit/2025-05-apebond-may-26th/issues/56

Vulnerability Detail

In Line 48 below, if there is a remaining SPL token balance in the treasury, it will send the SPL token to the `to: ctx.accounts.authority.to_account_info()`.

However, the issue here is that `ctx.accounts.authority` is a Signer, which means it is likely a wallet, and not a token account.

SPL token can only be sent to a token account. Thus, when the following code attempts to transfer SPL tokens to a non-token account address, it will revert.

```
File: close_treasury.rs
08: #[derive(Accounts)]
09: pub struct CloseTreasury<'info> {
10:     #[account(mut)]
11:     pub bond_issuance: Account<'info, BondIssuance>,
12:
13:     #[account(mut)]
14:     pub authority: Signer<'info>,
15:
16:     #[account(
17:         seeds = [b"permissioned_account", authority.key().as_ref()],
18:         bump,
19:         constraint = authority_permissions.has_admin_or_bond_creator_access() @
  ↪ ApeBondError::InvalidAuthority
20:     )]
21:     pub authority_permissions: Account<'info, PermissionedAccount>,
22:
23:     #[account(
24:         mut,
25:         associated_token::mint = bond_issuance.payout_mint,
26:         associated_token::authority = bond_issuance,
27:     )]
28:     pub treasury_ata: Account<'info, TokenAccount>,
29:
30:     pub token_program: Program<'info, Token>,
31: }
32:
33: pub fn close_treasury(ctx: Context<CloseTreasury>) -> Result<()> {
34:
35:     require!(
36:         matches!(ctx.accounts.bond_issuance.status, BondStatus::Closed),
37:         ApeBondError::BondIssuanceNotClosed
```

```
    );


    let treasury_balance = ctx.accounts.treasury_ata.amount;
    if treasury_balance > 0 {
        anchor_spl::token::transfer(
            CpiContext::new_with_signer(
                ctx.accounts.token_program.to_account_info(),
                anchor_spl::token::Transfer {
                    from: ctx.accounts.treasury_ata.to_account_info(),
                    to: ctx.accounts.authority.to_account_info(),
                    authority: ctx.accounts.bond_issuance.to_account_info(),
                },
                &[&[
                    b"bond_issuance",
                    ctx.accounts.bond_issuance.payout_mint.as_ref(),
                    ctx.accounts.bond_issuance.issuance_counter.to_le_bytes().as_ref(),
                    &[ctx.accounts.bond_issuance.bump],
                ]],
            ),
            treasury_balance,
        )?;
    }
```

Impact

Treasury cannot be closed, and the remaining SPL tokens will be struck in the program.

Code Snippet

https://github.com/sherlock-audit/2025-05-apebond-may-26th/blob/633b78dc37b158832f43896f7253166013df40b5/apebond-solana/programs/apebond/src/instructions/close_treasury.rs#L48

Recommendation

Consider transferring the remaining SPL tokens to the authority's ATA.

Issue H-3: Incorrect scaling of the debt_ratio

Source: https://github.com/sherlock-audit/2025-05-apebond-may-26th/issues/57

Vulnerability Detail

Currently, the formula for the `debt_ratio` is not correctly implemented because of the numerator not being multiplied by `1e18` scaling factor leading to subsequent rounding issues.

Impact

Rounding issues when price is close to zero.

Code Snippet

Let's take a look at the `debt_ratio` calculation process:

https://github.com/sherlock-audit/2025-05-apebond-may-26th/blob/main/apebond-solana/programs/apebond/src/processing/price_calculations.rs#L86-88

```
let numerator = current_debt_u128
        .checked_mul(payout_decimals_multiplier)
        .ok_or::<Error>(ApeBondError::ArithmeticOverflow.into())?;
```

As we can see here, the `numerator` is calculated by multiplying `current_debt` (principal decimals) by `payout_decimals_multiplier` (payout decimals). In comparison, the same value is calculated by multiplying on additional `1e18` scaling factor in the Solidity implementation of the same functionality:

```
debtRatio_ = (currentDebt() * 10 ** payoutToken.decimals() * 1e18) /
↪  payoutTokenInitialSupply;
```

```
@dev scaled by 1e18 to support 6 decimal principal token and debt. This avoids
↪  issues with rounding
```

However, this is not done in the Solana implementation.

Consider the following example:

```
debt_ratio = current_debt_u128 * payout_decimals_multiplier / initial_supply

price = (control_variable * debt_ratio) / principal_decimals_multiplier
```

Assume `control_variable = 1` so we can skip it:

```
price = debt_ratio / principal_decimals_multiplier
price = (current_debt_u128 * payout_decimals_multiplier / initial_supply) /
 ↪  principal_decimals_multiplier
price = (current_debt_u128 * payout_decimals_multiplier) / (initial_supply *
 ↪  principal_decimals_multiplier)
```

Here, the `current_debt_u128` variable is denominated in `principal_decimals_multiplier`, while initial_supply is denominated in `payout_decimals_multiplier`.

As the all decimals are mutually removed, this leads to a situation where the price can't be represented when it has a value of `0.5`, for instance, because of the following rounding down.

Tool Used

Manual Review

Recommendation

Consider aligning with the math from the Solidity version of the repo. Additionally, we also need to account for the `amount_to_calculate` in this case because if the price is in 18 decimals and, let's say, the `net_amount = 100e6` then we'd divide it by 18 decimals and will not get the correct amount in principal token decimals because `amount_to_calculate` is in principal decimals and we then convert it to the payout token decimals to get the payout.

Discussion

Doublo54

The issue is valid, but scaling by 1e18 isn't feasible. Initially, the code followed the Solidity implementation and used 1e18 scaling. However, this led to overflows on `u64` and caused the code to break. As a result, the current implementation uses 1e9 scaling to avoid the overflow. Any other solution?

rodiontr

> The issue is valid, but scaling by 1e18 isn't feasible. Initially, the code followed the Solidity implementation and used 1e18 scaling. However, this led to overflows on `u64` and caused the code to break. As a result, the current implementation uses 1e9 scaling to avoid the overflow. Any other solution?

that's a good question, let me think about it a bit

rodiontr

and can you please tell where such overflows would happen?

Issue M-1: Initialization of bond issuance can be DOSed

Source: https://github.com/sherlock-audit/2025-05-apebond-may-26th/issues/48

Vulnerability Detail

It was found that the initialization of bond issuance can be DOSed or blocked by malicious users.

The `init` constraint below will fail when an account already exists. In Solana, anyone can create an ATA account for any authority

https://github.com/sherlock-audit/2025-05-apebond-may-26th/blob/633b78dc37b158832f43896f7253166013df40b5/apebond-solana/programs/apebond/src/instructions/initialize_bond_issuance.rs#L56

```
File: initialize_bond_issuance.rs
50:     #[account(
51:         init,
52:         payer = authority,
53:         associated_token::mint = payout_mint,
54:         associated_token::authority = bond_issuance
55:     )]
56:     pub treasury_ata: Account<'info, TokenAccount>,
```

Assume that the payout_mint = USDC, and the current issuance_counter = 2. The creation of `treasury_ata` ATA depends on the following two (2) items:

- payout_mint,
- issuance_counter

Thus, malicious users can pre-compute the `treasury_ata` ATA address for payout_mint = USDC and issuance_counter = 3, and initialize the ATA.

From this point onwards, the `initialize_bond_issuance` will be blocked when the admin calls it because internally, it will always use payout_mint = USDC and issuance_counter = 3 to create the `treasury_ata` ATA address.

Impact

Initialization of bond issuance can be DOSed or blocked

Code Snippet

https://github.com/sherlock-audit/2025-05-apebond-may-26th/blob/633b78dc37b158832f43896f7253166013df40b5/apebond-solana/programs/apebond/src/instructions/initialize_bond_issuance.rs#L56

Recommendation

Use `init_if_needed` instead of `init` for creating ATA.

Issue M-2: Protocol roles are incorrectly assigned in several places

Source: https://github.com/sherlock-audit/2025-05-apebond-may-26th/issues/52

Vulnerability Detail

The problem is that the protocol does not assign roles for certain instructions breaking the compatibility with the documentation.

Impact

Certain roles cannot call the instructions they're supposed to call.

Code Snippet

In the migration documentation, there is a table of roles where it's written what function is supposed to be called and by whom. However, the following functions are deviated currently:

`set_min_price` - has has_admin_or_operator_or_automation_access(). But in the table it's only admin or operator:

https://github.com/sherlock-audit/2025-05-apebond-may-26th/blob/main/apebond-solana/programs/apebond/src/instructions/set_min_price.rs#L27

```
constraint = authority_permissions.has_admin_or_operator_or_automation_access() @
↪  ApeBondError::InvalidAuthority
```

`grant_role` and `revoke_roles()` - has the following constraint `constraint = authority_permissions.is_admin()` but in the table an admin or operator can grant and revoke the bond creator role:

https://github.com/sherlock-audit/2025-05-apebond-may-26th/blob/main/apebond-solana/programs/apebond/src/instructions/permissioned_account.rs#L14

```
constraint = authority_permissions.is_admin() @ ApeBondError::InvalidAuthority
```

Tool Used

Manual Review

Recommendation

Assign the roles in accordance with the documentation.

Discussion

Doublo54

Good catch!

- For set_min_price this was wrongly documented and automation role should also be able to call this method. Current code is good
- grant and revoke the bond creator role issue is valid it seems. Operations role should be able to grant and revoke bond creator role

Issue M-3: Inconsistency when calling payoutsAtTime() and claimable_payout()

Source: https://github.com/sherlock-audit/2025-05-apebond-may-26th/issues/55

Vulnerability Detail

In its essence, the `payoutsAtTime()` function is similar to the `claimable_payout()` and it just returns two additional parameters - `vested_payout` and `vesting_payout`. The problem is that these functions would return 2 different claimable payouts if there were already claimed amounts on behalf of the user.

Impact

Incorrect claimable payout being returned.

Code Snippet

Let's take a look at how the claimable payout is calculated inside of the `claimable_payout()` in the `claim_calculations`:

https://github.com/sherlock-audit/2025-05-apebond-may-26th/blob/main/apebond-solana/programs/apebond/src/processing/claim_calculations.rs#L49-69

```
// Calculate available payout based on vesting strategy
        match bond_term.vesting_strategy {
            0 => { // Linear vesting
                let payout = bond
                    .payout
                    .checked_mul(time_since_last_claim)
                    .ok_or(ApeBondError::ArithmeticOverflow)?
                    .checked_div(bond.vesting_term)
                    .ok_or(ApeBondError::ArithmeticOverflow)?;

                Ok(payout)
            },
            1 => { // Cliff vesting
                if current_timestamp >= vesting_end {
                    Ok(bond.payout - bond.payout_claimed)
                } else {
                    Ok(0)
                }
            },
            _ => Err(ApeBondError::InvalidVestingStrategy.into())
        }
```

As you can see here, the payout is calculated based on the `vesting_strategy` type - if it's linear, it's calculated linearly based on the elapsed time since the last claim and if it's a cliff vesting then the payout is given to the user after the `vestind_end` has been reached. On the other side, the `payouts_at_time()` has quite similar functionality:

https://github.com/sherlock-audit/2025-05-apebond-may-26th/blob/main/apebond-solana/programs/apebond/src/processing/claim_calculations.rs#L97-111

```
// Calculate vested payout until timestamp
    let vested_payout = match bond_term.vesting_strategy {
        0 => { // Linear vesting
            bond
                .payout
                .checked_mul(elapsed_time)
                .ok_or(ApeBondError::ArithmeticOverflow)?
                .checked_div(bond.vesting_term)
                .ok_or(ApeBondError::ArithmeticOverflow)?
        },
        1 => { // Cliff vesting
            0 // In cliff vesting, no tokens are unlocked until the end
        },
        _ => return Err(ApeBondError::InvalidVestingStrategy.into())
    };
```

The problem is that after that the `vested_payout` (at this point it's similar to what we have in the `claimable_payout()` function) is corrected based on the amount that has already been claimed:

https://github.com/sherlock-audit/2025-05-apebond-may-26th/blob/main/apebond-solana/programs/apebond/src/processing/claim_calculations.rs#L117-121

```
let claimable_payout = if vested_payout > bond.payout_claimed {
          vested_payout - bond.payout_claimed
      } else {
          0
      };
```

It can be seen that the `claimable_payout` depends on the already claimed payout. And, let's say, we claimed 5 tokens before, and now after some time we can claim 10 tokens that were linearly vested but as 10 tokens > 5 tokens, it returns only 5 tokens as the `claimable_payout` which is not correct claimable payout (at least with the linear vesting)

Tool Used

Manual Review

Recommendation

Consider having the same `claimable_payout` returned for the linear vesting in the `payouts_at_time()` function as in the `claimable_payout` function itself.

Issue L-1: Price slippage check differs in Solana & EVM

Source: https://github.com/sherlock-audit/2025-05-apebond-may-26th/issues/50

Vulnerability Detail

It was observed that there are some discrepancies in how the price slippage check is performed between the EVM and Solana implementations.

In the Solana implementation, the `true_price` does not take into account the fee.

```
File: deposit.rs
    // Calculate true price first
    let true_price = true_bond_price(
        &ctx.accounts.bond_pricing,
        &ctx.accounts.bond_term,
        current_timestamp,
        ctx.accounts.bond_issuance.principal_mint_decimals,
        ctx.accounts.bond_issuance.payout_mint_decimals,
    )?;
```

```
File: price_calculations.rs
/// Calculates the true bond price
pub fn true_bond_price(
    bond_pricing: &BondPricing,
    bond_term: &BondTerm,
    current_timestamp: u64,
    principal_mint_decimals: u8,
    payout_mint_decimals: u8,
) -> Result<u64> {
    let control_variable = bond_term.control_variable as u128;

    let debt_ratio = debt_ratio(
        bond_pricing.total_debt,
        bond_pricing.last_decay,
        current_timestamp,
        bond_term.vesting_end,
        payout_mint_decimals,
        bond_term.payout_token_initial_supply,
    )?;

    let numerator = control_variable
        .checked_mul(debt_ratio)
        .ok_or::<Error>(ApeBondError::ArithmeticOverflow.into())?;

    let price = numerator
```

```
        .checked_div(10u128.pow(principal_mint_decimals as u32))
        .ok_or::<Error>(ApeBondError::ArithmeticOverflow.into())?;

    // convert to u64 and check overflow
    let price_u64: u64 = price.try_into().map_err(|_| Error::from(ApeBondError::ArithmeticOverflow))?;

    let final_price = if price_u64 < bond_term.minimum_debt {
        bond_term.minimum_debt
    } else {
        price_u64
    };

    msg!("TrueBondPrice: price={}, debt_ratio={}", final_price, debt_ratio);
    Ok(final_price)
}
```

Then the price slippage is checked against the `true_price` (without fee)

```
File: deposit.rs
    pub fn validate(&self, max_price: u64, true_price: u64, new_total_debt: u64, total_payout: u64) -> Result<()> {
..SNIP..
        // Validate against max price
        require!(
            true_price <= max_price,
            ApeBondError::SlippageExceeded
        );
```

However, in EVM implementation, the fee is taken into account when checking the slippage.

```
File: CustomBill.sol
    /**
     * @notice calculate true bill price a user pays including the fee
     * @return price_ uint
     */
    function trueBillPrice() public view returns (uint256 price_) {
        price_ = (billPrice() * MAX_FEE) / (MAX_FEE - currentFee());
    }
```

```
File: CustomBill.sol
    function deposit(
        uint256 _amount,
        uint256 _maxPrice,
        address _depositor
    ) external nonReentrant returns (uint256) {
        require(_depositor != address(0), "Invalid address");
```

```
        require(msg.sender == _depositor || AddressUpgradeable.isContract(msg.sender), "no deposits to other address");

        _decayDebt();
        uint256 truePrice = trueBillPrice();
        require(_maxPrice >= truePrice, "Slippage more than max price"); // slippage protection
```

Impact

The program might not behave as expected.

Code Snippet

https://github.com/sherlock-audit/2025-05-apebond-may-26th/blob/633b78dc37b158832f43896f7253166013df40b5/apebond-solana/programs/apebond/src/instructions/deposit.rs#L148

Recommendation

Review both implementations to check if they adhere to the expected design.

Issue L-2: Token-2022 is not supported

Source: https://github.com/sherlock-audit/2025-05-apebond-may-26th/issues/51

This issue has been acknowledged by the team but won't be fixed at this time.

Vulnerability Detail

It was observed that many parts of the codebase support only legacy SPL tokens.

1. The following declaration only supports the legacy SPL Token program.

```
File: claim.rs
    pub token_program: Program<'info, Token>,
```

2. `anchor_spl::token::transfer` might break when used with Token-2022 token.

```
File: claim.rs
    pub fn handle_transfer(&self, claimable_payout: u64) -> Result<()> {
        anchor_spl::token::transfer(
            CpiContext::new_with_signer(
                self.token_program.to_account_info(),
                anchor_spl::token::Transfer {
                    from: self.treasury_ata.to_account_info(),
                    to: self.user_payout_ata.to_account_info(),
                    authority: self.bond_issuance.to_account_info(),
                },
                &[&[
                    b"bond_issuance",
                    self.bond_issuance.payout_mint.as_ref(),
                    self.bond_issuance.issuance_counter.to_le_bytes().as_ref(),
                    &[self.bond_issuance.bump],
                ]],
            ),
            claimable_payout,
        )?;
        Ok(())
    }
```

Impact

If the protocol intends to support both legacy SPL and newer Token-2022 standards, it will not work as expected.

Code Snippet

https://github.com/sherlock-audit/2025-05-apebond-may-26th/blob/633b78dc37b158832f43896f7253166013df40b5/apebond-solana/programs/apebond/src/instructions/claim.rs#L123

Recommendation

Consider using `anchor_spl::token_interface::transfer_checked`, which will work for both standards.

Issue L-3: Excessive ATA checking leads to excessive CU consumption

Source: https://github.com/sherlock-audit/2025-05-apebond-may-26th/issues/53

Vulnerability Detail

At the moment, the payer ATA is unnecessarily extensively checked in the `deposit` instruction making the function execution computationally more expensive.

Impact

Redundant CU consumption.

Code Snippet

Let's take a look at the following functionality:

https://github.com/sherlock-audit/2025-05-apebond-may-26th/blob/main/apebond-solana/programs/apebond/src/instructions/deposit.rs#L82-88

```
// Token accounts for transfers
 #[account(
     mut,
     associated_token::mint = bond_issuance.principal_mint,
     associated_token::authority = payer,
 )]
 pub payer_principal_ata: Box<Account<'info, TokenAccount>>,
```

Here, the `payer_principal_ata` account (the associated token account for the payer for the principal token) is checked for the `principal_mint` and the `authority`. Now let's take a look where the principal token is actually transferred:

https://github.com/sherlock-audit/2025-05-apebond-may-26th/blob/main/apebond-solana/programs/apebond/src/instructions/deposit.rs#L222-234

```
if principal_fee_amount > 0 {
            anchor_spl::token::transfer(
                CpiContext::new(
                    self.token_program.to_account_info(),
                    anchor_spl::token::Transfer {
                        from: self.payer_principal_ata.to_account_info(),
                        to: self.fee_principal_recipient_ata.to_account_info(),
                        authority: self.payer.to_account_info(),
                    },
                ),
```

```
            principal_fee_amount,
        )?;
    }
```

https://github.com/sherlock-audit/2025-05-apebond-may-26th/blob/main/apebond-solana/programs/apebond/src/instructions/deposit.rs#L237-247

```
anchor_spl::token::transfer(
        CpiContext::new(
            self.token_program.to_account_info(),
            anchor_spl::token::Transfer {
                from: self.payer_principal_ata.to_account_info(),
                to: self.partner_principal_recipient_ata.to_account_info(),
                authority: self.payer.to_account_info(),
            },
        ),
        principal_billed_amount,
    )?;
```

But both of the destination accounts - `fee_principal_recipient_ata` and `partner_principal_recipient_ata` are also similarly checked in the `Deposit` struct:

https://github.com/sherlock-audit/2025-05-apebond-may-26th/blob/main/apebond-solana/programs/apebond/src/instructions/deposit.rs#L90-95

```
#[account(
    mut,
    associated_token::mint = bond_issuance.principal_mint,
    associated_token::authority = bond_issuance.fee_principal_recipient,
)]
pub fee_principal_recipient_ata: Box<Account<'info, TokenAccount>>,
```

https://github.com/sherlock-audit/2025-05-apebond-may-26th/blob/main/apebond-solana/programs/apebond/src/instructions/deposit.rs#L97-102

```
#[account(
       mut,
       associated_token::mint = bond_issuance.principal_mint,
       associated_token::authority = bond_issuance.partner_principal_recipient,
   )]
   pub partner_principal_recipient_ata: Box<Account<'info, TokenAccount>>,
```

The problem is that we don't actually need to check the same way both the source ATA and the destination ATA as he Token program internally checks:

- the owner of the ATA has to be a signer for the transfer so if the payer is not the signer for the transfer, the tx would fail
- the mint of the source and destination matches so there is no need to check the mint on both sides

Tool Used

Manual Review

Recommendation

The checks for the `payer_principal_ata` can be removed:

```
pub payer_principal_ata: Box<Account<'info, TokenAccount>>,
```

Same thing also applies for this transfer of the payout token from the treasury ATA:

https://github.com/sherlock-audit/2025-05-apebond-may-26th/blob/main/apebond-solana/programs/apebond/src/instructions/deposit.rs#L251-269

```
if payout_fee_amount > 0 {
            anchor_spl::token::transfer(
                CpiContext::new_with_signer(
                self.token_program.to_account_info(),
                anchor_spl::token::Transfer {
                    from: self.treasury_ata.to_account_info(),
                    to: self.payout_fee_recipient_ata.to_account_info(),
                    authority: self.bond_issuance.to_account_info(),
                },
                &[&[
                    b"bond_issuance",
                    self.bond_issuance.payout_mint.as_ref(),
                    self.bond_issuance.issuance_counter.to_le_bytes().as_ref(),
                    &[self.bond_issuance.bump],
                ]],
            ),
                payout_fee_amount,
            )?;
        }
```

Issue L-4: Insecure initialization

Source: https://github.com/sherlock-audit/2025-05-apebond-may-26th/issues/54

This issue has been acknowledged by the team but won't be fixed at this time.

Vulnerability Detail

It was observed that the `initialize_permissioned_account()` function is not gated. As a result, anyone can trigger the `initialize_permissioned_account()` and take control of the program if the protocol team hasn't executed `initialize_permissioned_account()` yet.

```
File: permissioned_account.rs
#[derive(Accounts)]
pub struct InitializePermissionedAccount<'info> {
    #[account(mut)]
    pub authority: Signer<'info>,

    #[account(
        init,
        payer = authority,
        space = PermissionedAccount::LEN,
        seeds = [b"permissioned_account", authority.key().as_ref()],
        bump
    )]
    pub authority_permissions: Account<'info, PermissionedAccount>,

    #[account(
        init,
        payer = authority,
        space = DefaultAdminSetup::LEN,
        seeds = [b"default_admin_setup"],
        bump
    )]
    pub admin_setup: Account<'info, DefaultAdminSetup>,

    pub system_program: Program<'info, System>,
}
```

```
File: permissioned_account.rs
pub fn initialize_permissioned_account(
    ctx: Context<InitializePermissionedAccount>,
) -> Result<()> {
    let authority_permissions = &mut ctx.accounts.authority_permissions;
    authority_permissions.roles = ROLE_ADMIN; // Initialize with admin role

    let admin_setup = &mut ctx.accounts.admin_setup;
```

```
    admin_setup.initialized = true;

    Ok(())
}
```

Impact

Anyone can take control of the program if the protocol team has not yet initialized it.

Code Snippet

https://github.com/sherlock-audit/2025-05-apebond-may-26th/blob/633b78dc37b158832f43896f7253166013df40b5/apebond-solana/programs/apebond/src/instructions/permissioned_account.rs#L111

Recommendation

Ensures that only the program's upgrade_authority can call the `initialize_permissioned_account()` function. Add the following to the `pub struct InitializePermissionedAccount`.

```
#[account(constraint = program.programdata_address()? == Some(program_data.key()))]
pub program: Program<'info, MyProgram>,
#[account(constraint = program_data.upgrade_authority_address == Some(authority.key()))]
pub program_data: Account<'info, ProgramData>,
```

The following is the rationale for the above constraints:

1. The first constraint on `program` ensures that the provided `program_data` account matches the program's `programdata_address` field.
2. The second constraint on the `program_data`account ensures that the instruction's signer matches the `program_data` account's `upgrade_authority_address` field.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.