FYEO

Security Code Review of PegaX

PegaX

August 2025
Version 0.9

Presented by:
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Security Level
Public

Table of Contents

Executive Summary

Overview

PegaX engaged FYEO Inc. to perform a Security Code Review of PegaX.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on July 31 - August 05, 2025, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following issues have been identified during the testing period. These should be prioritized for remediation to reduce the risk they pose:

- FYEO-PEGAX-01 – Incorrect token accounting
- FYEO-PEGAX-02 – Insecure ed25519 Parsing
- FYEO-PEGAX-03 – General improvements
- FYEO-PEGAX-04 – Incomplete Validation of Operator and Verifier Keys
- FYEO-PEGAX-05 – Insecure initialisation
- FYEO-PEGAX-06 – Missing bounds check
- FYEO-PEGAX-07 – Potential improper ATA validation with changed authority

Based on our review process, we conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

The FYEO Review Team performed a Security Code Review of PegaX. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a private repository at
https://github.com/PegaxTrade/pegax-program with the commit hash
771ba91aaa008b2561d4497f6d308039bc48ddf0.

Remediations were submitted with the commit hash 17aac228bd8075c55e6b41e13ebb00935ddffca1.

Files included in the code review

```
pegax-program/
└── programs/
    └── pegax/
        └── src/
            ├── admins/
            │   ├── admin_community_auth_update.rs
            │   ├── admin_fee_update.rs
            │   ├── admin_fee_withdraw.rs
            │   ├── admin_operator_transfer.rs
            │   ├── admin_operator_update.rs
            │   ├── admin_verifier_transfer.rs
            │   ├── admin_verifier_update.rs
            │   ├── create_metadata_state.rs
            │   └── mod.rs
            ├── clients/
            │   ├── mod.rs
            │   ├── pump.rs
            │   ├── pump_amm.rs
            │   ├── raydium.rs
            │   ├── raydium_cpmm.rs
            │   └── raydium_launchpad.rs
            ├── instructions/
            │   ├── mod.rs
            │   ├── pump_amm_buy.rs
            │   ├── pump_amm_sell.rs
            │   ├── pump_buy.rs
            │   ├── pump_sell.rs
            │   ├── raydium_cpmm_swap.rs
            │   ├── raydium_launchpad_buy.rs
            │   ├── raydium_launchpad_sell.rs
            │   ├── raydium_swap.rs
```

Files included in the code review

```
│   ├── user_withdraw_sol.rs
│   └── user_withdraw_spl.rs
├── states/
│   ├── metadata_state.rs
│   ├── mod.rs
│   └── token_auth.rs
├── utils/
│   ├── account.rs
│   ├── constant.rs
│   ├── cpi.rs
│   ├── ed25519.rs
│   ├── events.rs
│   ├── fee.rs
│   ├── hash.rs
│   ├── math.rs
│   ├── mod.rs
│   ├── parameter.rs
│   ├── swap_context.rs
│   └── validate.rs
├── error.rs
└── lib.rs
```

Table 1: Scope

Technical Analyses and Findings

During the Security Code Review of PegaX, we discovered:

- 2 findings with MEDIUM severity rating.
- 5 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

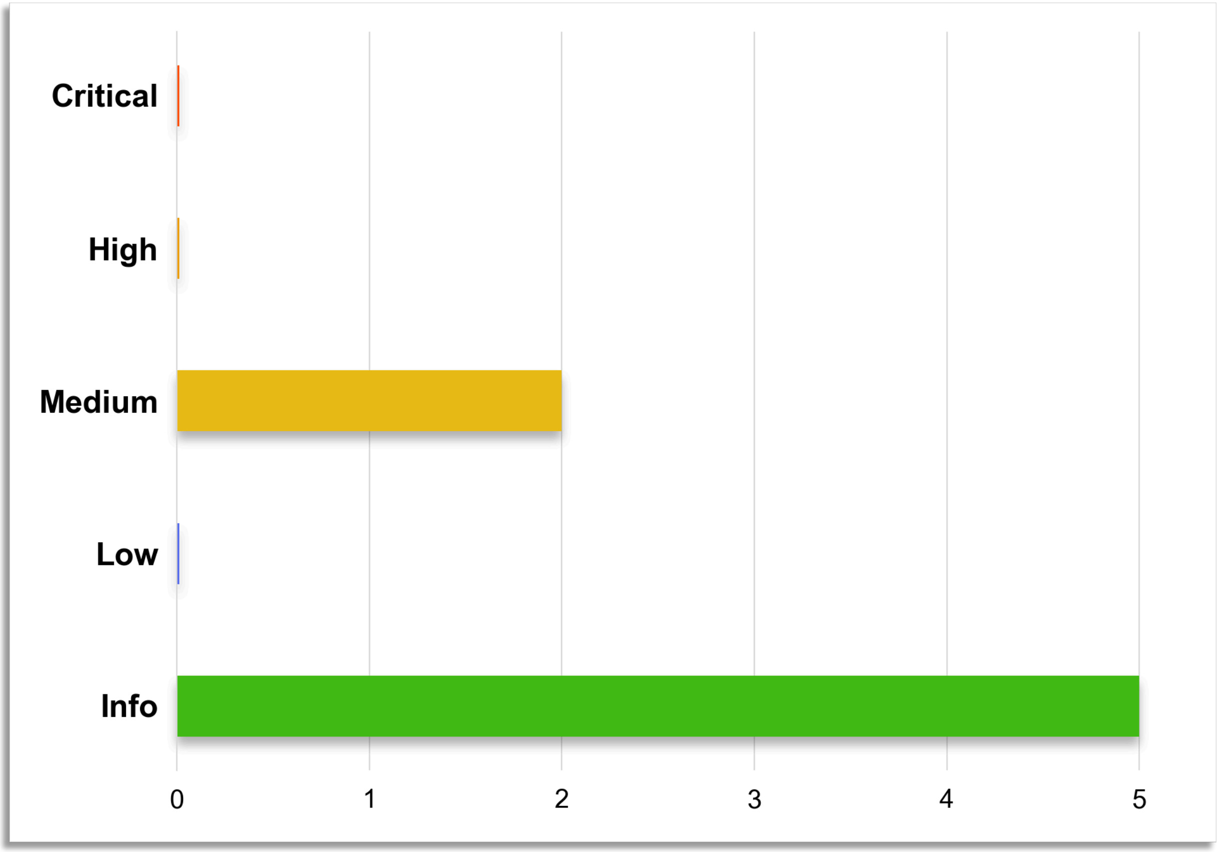

Figure 1: Findings by Severity

Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

Finding #	Severity	Description	Status
FYEO-PEGAX-01	Medium	Incorrect token accounting	Remediated
FYEO-PEGAX-02	Medium	Insecure ed25519 Parsing	Remediated
FYEO-PEGAX-03	Informational	General improvements	Remediated
FYEO-PEGAX-04	Informational	Incomplete Validation of Operator and Verifier Keys	Remediated
FYEO-PEGAX-05	Informational	Insecure initialisation	Remediated
FYEO-PEGAX-06	Informational	Missing bounds check	Remediated
FYEO-PEGAX-07	Informational	Potential improper ATA validation with changed authority	Remediated

Table 2: Findings Overview

The Classification of vulnerabilities

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets

- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations

Technical Analysis

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

Conclusion

Based on our review process, we conclude that the code implements the documented functionality to the extent of the reviewed code.

Technical Findings

General Observations

This Solana program (“PegaX”) functions as a configurable on-chain swap hub that lets users seamlessly exchange SOL and SPL tokens across multiple liquidity sources. At its core, it maintains a shared metadata state governed by designated “community,” “operator,” and “verifier” roles that defines swap parameters, fee schedules, and authorized signers. Users submit signed swap orders (for example, buying or selling through Pump.fun’s AMM or Raydium’s pools and launchpads), and the contract atomically routes their funds, applies platform/community operator fees, and emits standardized swap events for downstream tracking.

In addition to swaps, the program equips administrators with a suite of governance tools: they can update the global metadata, adjust fee percentages and fee recipients, reassign operator and verifier roles, and withdraw accumulated fees. End users also have simple withdrawal endpoints to claim any SOL or SPL balances held under their dedicated on-chain vault (PDA-derived escrow). Overall, Pegax blends permissioned governance with multi-venue swap execution and transparent fee distribution, offering a turnkey DeFi primitive on Solana.

Incorrect token accounting

Finding ID: FYEO-PEGAX-01
Severity: Medium
Status: Remediated

Description

The `RaydiumLaunchpadBuy` and `RaydiumLaunchpadSell` instructions use a token accounts lamport balance instead of the token balance to account for how many tokens were swapped.

Proof of Issue

File name: programs/pegax/src/instructions/raydium_launchpad_buy.rs
Line number: 224

```
let pool_vault_in_before = ctx.accounts.pool_vault_in.get_lamports();

cpi_wrap_sol(&ctx, &ctx.accounts.in_token_program, &ctx.accounts.user_token_auth_ata_in,
wrap_amount)?;
cpi_buy(&ctx, parameters.in_amount_max, parameters.out_amount)?;
cpi_unwrap_sol(&ctx, &ctx.accounts.in_token_program,
&ctx.accounts.user_token_auth_ata_in)?;

let pool_vault_in_after = ctx.accounts.pool_vault_in.get_lamports();

let swap_amount = ValueU64::from(pool_vault_in_after).sub(pool_vault_in_before)?.get();
```

Reload the account to get the updated token balance.

Severity and Impact Summary

The `swap_amount` will be incorrect.

Recommendation

Make sure to correctly calculate the swapped amount in both instructions. Add tests to verify correct fee calculations.

Insecure ed25519 Parsing

Finding ID: FYEO-PEGAX-02
Severity: Medium
Status: Remediated

Description

The `parse_ed25519` function assumes that the instruction data is in a specific format and directly slices the data without checking if the data is valid. This could lead to out-of-bounds memory access or incorrect parsing of the public key and message.

Proof of Issue

File name: programs/pegax/src/utils/ed25519.rs
Line number: 63

```
fn parse_ed25519(instruction: &Instruction) -> Result<(Pubkey, [u8; 32])> {
    let data = &instruction.data;

    let public_key: [u8; 32] = data[16..16 + 32].try_into().unwrap();
    let message: [u8; 32] = data[112..112 + 32].try_into().unwrap();

    Ok((
        Pubkey::from(public_key),
        message,
    ))
}
```

Severity and Impact Summary

These offsets are assumed to be correct, the data could potentially be shifted.

Recommendation

Verify the offsets are correct as declared in:

```
message_data_offset (bytes 10-11)
message_data_size (bytes 12-13)
```

General improvements

Finding ID: FYEO-PEGAX-03
Severity: Informational
Status: Remediated

Description

Some code optimisations that could be done to improve the maintainability of the codebase.

Proof of Issue

File name: programs/pegax/src/states/metadata_state.rs
Line number: 29

```
pub fn is_operator(&self, public_key: &Pubkey) -> bool {
    self.operators.iter()
        .filter(|operator| *operator != &Pubkey::default())
        .any(|operator| operator == public_key)
}
```

Fail early if `public_key == Pubkey::default()`. Otherwise use simple `any()` test.

File name: programs/pegax/src/utils/ed25519.rs
Line number: 11

```
pub fn verify_ed25519_all<'info>(
    sysvar_instructions: &AccountInfo<'info>,
    hash_bytes: &Vec<u8>,
    verifiers: &[Pubkey],
    verifier_count: u16,
) -> Result<()> {
    let verifiers = (1..=verifier_count
```

The `verifiers` could be renamed to `allowed_verifiers`. Furthermore the variable is immediately shadowed.

File name: programs/pegax/src/utils/ed25519.rs
Line number: 38

```
fn verify_ed25519<'info>(
    sysvar_instructions: &AccountInfo<'info>,
    hash_bytes: &Vec<u8>,
    verifiers: &[Pubkey],
```

The `verifiers` could be renamed to `allowed_verifiers`.

Severity and Impact Summary

Not a security concern.

Recommendation

Improving the maintainability of the codebase helps to avoid potential issues later on.

Incomplete Validation of Operator and Verifier Keys

Finding ID: FYEO-PEGAX-04
Severity: Informational
Status: Remediated

Description

The code uses constraints to validate the operators and verifiers, but it does not ensure that the keys are not duplicated. This could lead to potential issues if duplicate keys are used.

Proof of Issue

File name: programs/pegax/src/utils/validate.rs
Line number: 17

```
pub fn valid_public_keys(public_keys: &[Pubkey]) -> bool {
    public_keys.iter().any(|public_key| public_key != &Pubkey::default())
}
```

This only requires one non-zero key.

Severity and Impact Summary

The code does not enforce uniqueness checks on the operators and verifiers. While the constraints validate that the keys are valid public keys, they do not ensure that the keys are unique within the arrays. This can store redundant data and give bad information about the number of current operators and verifiers.

Recommendation

Make sure to not allow duplicates when storing these data.

Insecure initialisation

Finding ID: FYEO-PEGAX-05
Severity: Informational
Status: Remediated

Description

The initialisation is done using a public function. There is a risk someone else can call it first.

Proof of Issue

File name: programs/pegax/src/admins/create_metadata_state.rs
Line number: 23

```
pub struct CreateMetadataState<'info> {
    #[account(
        init,
        payer = operator_admin,
        space = 8 + MetadataState::INIT_SPACE + 1024,
        seeds = [MetadataState::METADATA_STATE_PDA_SEED],
        bump,
        constraint = validate::valid_public_keys(&parameters.operators)
@ErrorCode::ErrorInvalidOperators,
        constraint = validate::valid_public_keys(&parameters.verifiers)
@ErrorCode::ErrorInvalidVerifiers,
        constraint = validate::valid_platform_fee_ratio(parameters.platform_fee_ratio)
@ErrorCode::ErrorInvalidPlatformFeeRatio,
        constraint =
validate::valid_total_fee_ratio(parameters.community_fee_sharing_ratio,
parameters.development_fee_sharing_ratio) @ErrorCode::ErrorInvalidTotalFeeRatio,
    )]
    pub metadata_state: Box<Account<'info, MetadataState>>,

    ...

    #[account(
        mut,
    )]
    pub operator_admin: Signer<'info>,
```

Severity and Impact Summary

Anyone can initialise this program.

Recommendation

For upgradeable programs, it is recommended to use the following pattern to initialise the program.

```
pub owner: Signer<'info>,

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

Missing bounds check

Finding ID: FYEO-PEGAX-06
Severity: **Informational**
Status: **Remediated**

Description

While it was stated that `verifier_count_min` can be 0, there is no check that the count is less than or equal to the length of `parameters.verifiers`. So an impossible threshold can be set.

Proof of Issue

File name: programs/pegax/src/admins/admin_verifier_update.rs
Line number: 35

```
pub fn admin_verifier_update(ctx: Context<AdminVerifierUpdate>, parameters:
AdminVerifierUpdateParameters) -> Result<()> {
    ctx.accounts.metadata_state.verifiers = parameters.verifiers;
    ctx.accounts.metadata_state.verifier_count_min = parameters.verifier_count_min;
```

Severity and Impact Summary

Out of bound values may prevent the program from operating correctly.

Recommendation

Set limitations on all configuration options.

Potential improper ATA validation with changed authority

Finding ID: FYEO-PEGAX-07
Severity: **Informational**
Status: **Remediated**

Description

Anchor's `associated_token::authority` constraint verifies that the ATA's owner PDA matches the expected authority. If the owner has ever called SPL SetAuthority on the token account (pre-Token2022), the 'authority' no longer equals the PDA and Anchor will reject the instruction at validation time.

Proof of Issue

File name: programs/pegax/src/instructions/user_withdraw_spl.rs
Line number: 48

```
#[account(
    mut,
    associated_token::mint = mint,
    associated_token::authority = recipient,
    associated_token::token_program = token_program,
)]
pub recipient_ata: Box<InterfaceAccount<'info, TokenAccount>>
```

Severity and Impact Summary

Any user who ever had their ATA's owner changed (often by some scam calling SetAuthority) will be unable to withdraw funds resulting in a Denial-of-Service for that user.

Recommendation

Consider if there should be another way to withdraw tokens that do not involve the ATA. The check could accept any token account with the correct `token::authority` set.

Our Process

Methodology

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

Kickoff
Ramp-up
Review
Report
Verify

Figure 2: Methodology Flow

Kickoff

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It’s an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.