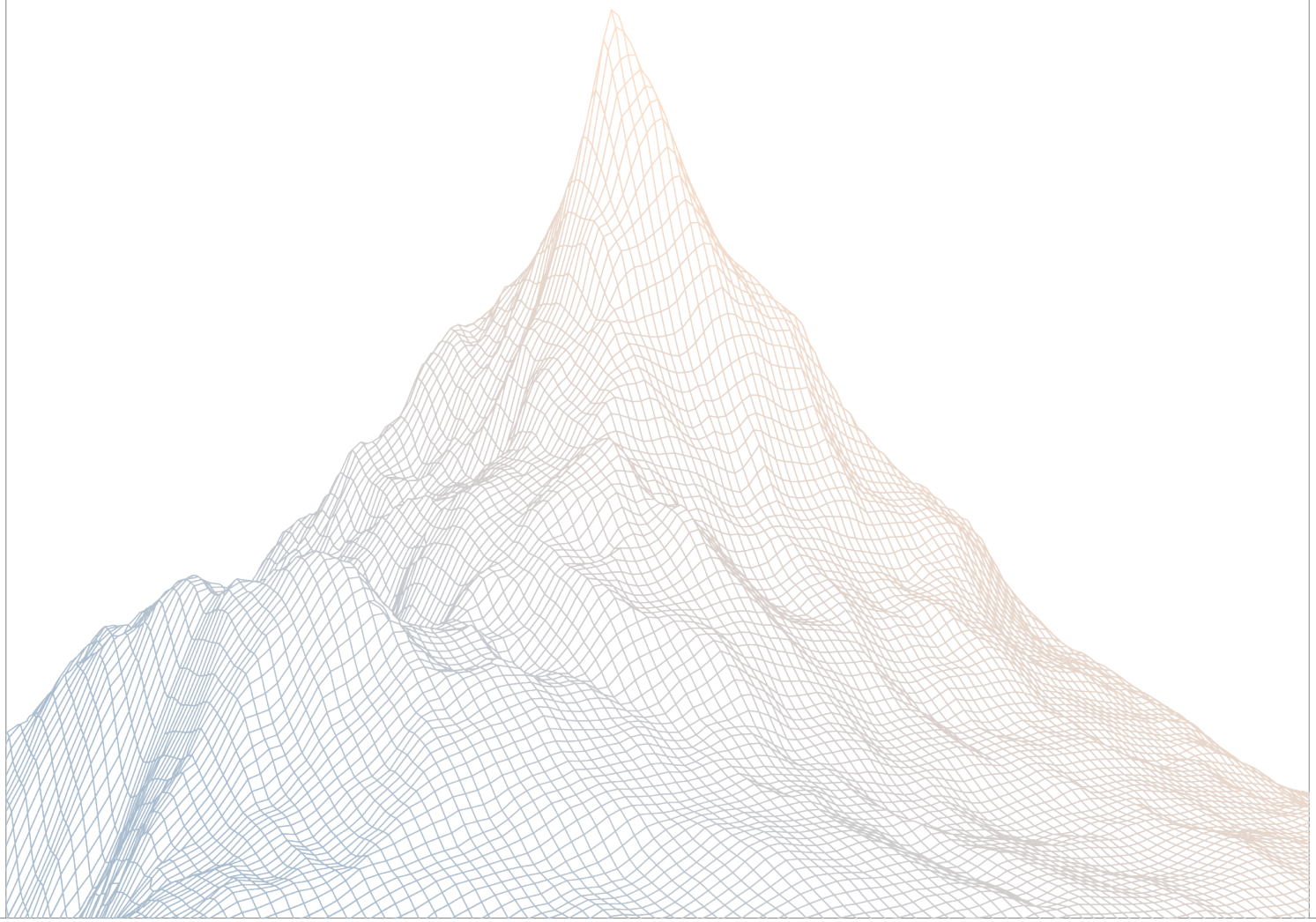
Zenith

Alien

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: December 15th to December 16th, 2025

AUDITED BY: peakbolt
shaflow

Contents

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2

Executive Summary

2.1 About Alien

Alien is a highly scalable network of unique real humans designed to enable trust in the age of AI. The Alien Network implements CHVP, which grants every person in the world a unique Alien ID. The system is fair launched, decentralized, and programmable to support identity, payments, and trust across every sector of the internet.

2.2 Scope

The engagement involved a review of the following targets:

Target	p2p-solana-contracts
Repository	https://github.com/alien-id/p2p-solana-contracts
Commit Hash	`c5026bce5c8164df725dfeb293d2df2dd3e88360`
Files	`programs/walien-pool/src/**/*.rs`

2.3 Audit Timeline

December 15, 2025	Audit start
December 16, 2025	Audit end
January 19, 2026	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	1
Medium Risk	2
Low Risk	5
Informational	4
Total Issues	**12**

3

Findings Summary

ID	Description	Status
H-1	Users may pay an excessive amount of USDC	Resolved
M-1	last buyer can be griefed by performing dust buy	Acknowledged
M-2	calculate_swap_from_config uses an incorrect target tick	Resolved
L-1	Lack of validation for usdc_mint could cause price assumption to be wrong	Resolved
L-2	Missing validation for set_walien() could cause mis-pricing	Resolved
L-3	Improper account creation checks may cause rent to be refunded to an incorrect address	Resolved
L-4	GlobalConfig does not set the fee parameter	Resolved
L-5	Missing price sanity checks	Resolved
I-1	Prevention of zero amount_out token purchases	Resolved
I-2	Redundant tick_upper field	Resolved
I-3	Implement a two-step ownership transfer	Acknowledged
I-4	Incorrect parameter naming	Resolved

4

Findings

4.1 High Risk

A total of 1 high risk findings were identified.

[H-1] Users may pay an excessive amount of USDC

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: Medium

Target

- buy.rs

Description:

When the `buy()` instruction calculates how much Walien can be obtained from the input USDC, if the price exceeds the target, the remaining available `amountOut` is used to reverse-calculate `amountIn`. In this case, the `amountIn` in the swap result will be less than the original `amount`.

```
pub fn apply(...) -> Result<()> {
    //...
    {
        let cpi_accounts = token::Transfer {
            from: ctx.accounts.user_usdc_ata.to_account_info(),
            to: ctx.accounts.program_usdc_token_account.to_account_info(),
            authority: ctx.accounts.user.to_account_info(),
        };
        let cpi_program = ctx.accounts.token_program.to_account_info();
        let transfer_ctx = CpiContext::new(cpi_program, cpi_accounts);

        token::transfer(transfer_ctx, amount)?;
    }
    //...
    Ok(())
}
```

However, during the actual user transfer, the program still uses the original `amount` instead of `amountIn`. This may cause users to pay an excessive amount of USDC.

Recommendations:

It is recommended to transfer `calculation_result.amount_in + calculation_resul.fee_amount` instead of `amount` during the transaction.

Alien: Resolved with `@9531238041 ...` and `@b8549ece6e ...`

Zenith: Verified.

4.2 Medium Risk

A total of 2 medium risk findings were identified.

[M-1] last buyer can be griefed by performing dust `buy`

SEVERITY: Medium	IMPACT: Medium
STATUS: Acknowledged	LIKELIHOOD: Medium

Target

- buy.rs#L102-L104

Description:

`buy()` is intended to accept a user's USDC payment, compute their WALIEN allocation, and decrement the remaining USDT budget for the sale.

The function updates `available_for_swap_in_usdt` using an unchecked `-=` subtraction, which can underflow when a prior "dust" buy reduces the remaining allocation below the next buyer's requested `amount`.

This enables an attacker to grief/DoS the final buyer by causing their `buy` to revert, preventing the sale from being completed as intended.

```
ctx.accounts
.global_config_account
.available_for_swap_in_usdt -= amount;
```

Recommendations:

Consider enforcing a minimum buy amount to make it more expensive for the attacker to perform such griefing attacks. (except in the case where the buy will consume the entire `available_for_swap_in_usdt`).

Alien: Acknowledged.

[M-2] `calculate_swap_from_config` uses an incorrect target tick

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- utils.rs

Description:

When a user calls the `quote()` instruction to get a price quote or the `buy()` instruction to purchase Walien, the `calculate_swap_from_config()` function is called to calculate how much Walien can be swapped from the input USDC. Within `calculate_swap_from_config()`, the `compute_swap()` function from `orca_math` is invoked to perform the swap calculation.

```
pub fn calculate_swap_from_config(cfg: &GlobalConfig, amount: u64) →
    Result<SwapStepComputation> {
    // ...
    let amount_specified_is_input = true;
    let a_to_b = true;
    Ok(compute_swap(
        amount,
        fee_rate,
        cfg.liqudity as u128,
        cfg.initial_sqrt_price_x64,
        sqrt_price_from_tick_index(cfg.tick_upper),
        amount_specified_is_input,
        a_to_b,
    )?)
}
```

However, `compute_swap()` is called with an incorrect `sqrt_price_target`. Since both `amount_specified_is_input` and `a_to_b` are set to `true`, the swap result will move the price downward. Therefore, `cfg.tick_lower` should be used as the price boundary instead of `cfg.tick_upper`. This can result in a situation where, if `initial_sqrt_price_x64` is initialized below `tick_upper`, the program will continue selling Walien even when the price is already below `tick_lower`, as long as `available_for_swap_in_usdt` remains.

Recommendations:

It is recommended to change `sqrt_price_target` to `cfg.tick_lower`.

Alien: Resolved with `@3a2baa7103 ...`

Zenith: Verified. Resolved by setting `a_to_b = false`.

4.3 Low Risk

A total of 5 low risk findings were identified.

[L-1] Lack of validation for `usdc_mint` could cause price assumption to be wrong

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- initialize.rs#L44-L68

Description:

`initialize()` sets up the global config and program USDC vault used for the sale/quote flow.

It accepts an arbitrary `usdc_mint` without validating it is the canonical USDC mint (and expected decimals), and only records whatever mint was provided into config.

A non-USDC mint (or wrong-decimals USDC variant) can be initialized, causing the initial price/quotes and accounting assumptions to be wrong and potentially enabling value loss or mispriced sales.

Recommendations:

Hardcode and enforce `usdc_mint == USDC mint pubkey` (and optionally `usdc_mint.decimals == 6`) during `initialize()` so price math/units remain consistent.

Alien: Resolved with `@fd9b1172ef ...`

Zenith: Verified. Resolved by verifying against `USDC_DECIMALS` (6).

[L-2] Missing validation for `set_walien()` could cause mis-pricing

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- set_walien.rs#L42-L46

Description:

`set_walien()` sets the WALIEN mint used by the pool and initializes the program's WALIEN vault account.

It does not validate that the provided WALIEN mint has the expected decimals or that it is consistent with the initial price scaling assumptions.

If a mint with unexpected decimals is set, quotes/buys/claims can be mispriced due to incorrect decimal normalization, leading to incorrect token allocations and value loss for users or the protocol.

Recommendations:

In `set_walien()`, enforce `walien_mint.decimals == expected_decimals` (and/or store expected decimals in config) and reject mints that don't match the initial price/decimal scaling assumptions used by the pool math.

Alien: Resolved with `@e81221042d ...`

Zenith: Verified. Resolved by checking `WALIEN_DECIMALS`.

[L-3] Improper account creation checks may cause rent to be refunded to an incorrect address

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- claim.rs

Description:

In the `claim()` instruction, the `user_position` account is closed. If the instruction caller paid the rent for the `user_walien_token_account`, the rent from `user_position` will be refunded to the caller. Otherwise, it will be refunded to the user.

```
pub fn apply(ctx: &mut Context<Claim>, _possition_index: u64) → Result<()>
    {
    //...
    let first_possition = ctx.accounts.user_walien_token_account.amount
    == 0;
    //...
    let recipient = if first_possition {
        ctx.accounts.caller.to_account_info()
    } else {
        ctx.accounts.user.to_account_info()
    };
    ctx.accounts.user_account.walien_allocation = 0;
    ctx.accounts.user_account.close(recipient)?;
    //...
    }
```

However, determining whether the rent was paid by the caller based on whether the `user_walien_token_account` token balance is zero is inappropriate. If the user has already created the ATA but has not transferred any tokens into it, or has transferred the received tokens out, the logic would still treat the rent as having been paid by the caller. This can result in the rent from `user_position` being incorrectly refunded to the caller.

Recommendations:

It is recommended not to use `init_if_needed` and the `TokenAccount` constraint. Instead, manually check in the handler whether the `user_walien_token_account` has already been created. If it has not been created, create it via CPI and refund the rent to the caller. Otherwise, refund the rent to the user.

Alien: Resolved with `@55082da8f9 ...`, `@8d8215e5fc ...`, `@919d205608 ...` and `@dc9dfa18bf ...`

Zenith: Verified.

[L-4] `GlobalConfig` does not set the fee parameter

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Medium

Target

- initialize.rs

Description:

The `GlobalConfig` account has a `fee_bps` field, which allows charging a fee proportionally when users purchase Walien.

```
pub fn apply(...) -> Result<()> {
    //...
    ctx.accounts.global_config_account.fee_bps = 0;
    //...
}
```

In the `Initialize()` instruction, `fee_bps` is not set, and there is no other instruction to adjust it. Therefore, the admin cannot configure the fee and cannot apply a fee when users buy Walien.

Recommendations:

It is recommended to pass and set `fee_bps` in the `Initialize()` instruction or add an instruction to update it.

Alien: Resolved with `@4851c9fc40...` and `@d57a3f31b1...`

Zenith: Verified.

[L-5] Missing price sanity checks

SEVERITY: Low	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Low

Target

- initialize.rs

Description:

The `initialize()` instruction sets the initial price for the Walien sale, determined by `tick_lower`, `tick_upper`, and `initial_sqrt_price_x64`.

```
pub fn apply(
    ctx: &mut Context<Initialize>,
    initial_sqrt_price_x64: u128,
    tick_lower: i32,
    tick_upper: i32,
    available_for_swap_in_usdt: u64,
    liquidity: u128,
) → Result<()> {
    // ...
    ctx.accounts.global_config_account.initial_sqrt_price_x64
    = initial_sqrt_price_x64;
    ctx.accounts.global_config_account.tick_lower = tick_lower;
    ctx.accounts.global_config_account.tick_upper = tick_upper;
    // ...
}
```

However, there is no sanity check on the price. Therefore, the three prices may exceed the maximum or minimum values, and `initial_sqrt_price_x64` could be outside the range of `tick_lower` and `tick_upper`, which may cause swap calculations to deviate from expectations.

Recommendations:

It is recommended to check that `tick_lower`, `tick_upper`, and `initial_sqrt_price_x64` are within the correct ranges, and that `tick_lower` < `initial_tick` < `tick_upper`.

Alien: Resolved with `@e0d438fd24 ...` and `@b944c66e94 ...`

Zenith: Verified.

4.4 Informational

A total of 4 informational findings were identified.

[I-1] Prevention of zero `amount_out` token purchases

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- claim.rs

Description:

In the `buy()` instruction, if the Walien price is higher than USDC, a too small USDC input may result in `amount_out` being 0. If the user has not set slippage control, a transaction with `amount_out` of 0 could still be executed. The user paid USDC and rent, but a `user_position` account with a `walien_allocation` of 0 was created.

```
pub fn apply(ctx: &mut Context<Claim>, _possition_index: u64) → Result<()>
    {
    // ...
    require!(
        ctx.accounts.user_account.walien_allocation > 0,
        ErrorCode::NothingToClaim
    );
    // ...
```

Moreover, in the `claim()` instruction, the `user_position` cannot be closed because `walien_allocation` is 0, resulting in a loss of rent.

Recommendations:

It is recommended to prevent purchases with `amount_out` of 0 in the `buy()` instruction.

Alien: Resolved with @cdd2d34715 ...

Zenith: Verified.

[I-2] Redundant `tick_upper` field

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- initialize.rs
- state.rs

Description:

The `GlobalConfig` account has `tick_lower` and `tick_upper` fields, and liquidity is added within this range.

```
pub fn apply(...) -> Result<()> {
    //...
    ctx.accounts.global_config_account.initial_sqrt_price_x64
    = initial_sqrt_price_x64;
    ctx.accounts.global_config_account.tick_lower = tick_lower;
    ctx.accounts.global_config_account.tick_upper = tick_upper;
    //...
}
```

However, during the actual Walien sale, the price only decreases, so only `tick_lower` is needed to limit the minimum price. `tick_upper` is unnecessary. Redundant fields will waste rent.

Recommendations:

It is recommended to remove the `tick_upper` field and only ensure that `tick_lower < initial_sqrt_price_x64`.

Alien: Resolved with `@19d38efa71 ...`

Zenith: Verified. Resolved by switching swap direction (`a_to_b = false`) and remove the now redundant `tick_lower`.

[I-3] Implement a two-step ownership transfer

SEVERITY: Informational	IMPACT: Informational
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- transfer_admin_authority.rs

Description:

In the `transfer_admin_authority()` instruction, the transfer of admin privileges does not verify the recipient.

```
pub fn apply(
    ctx: &mut Context<TransferAdminAuthority>,
    new_admin_authority: Pubkey,
) → Result<()> {
    ctx.accounts.global_config_account.admin = new_admin_authority;
    Ok(())
}
```

If `new_admin_authority` is set to an incorrect address, the project team may lose control of the program, becoming unable to set `is_claim_active` and `is_sale_active`, and potentially unable to call `deposit_walien()` to collect funds. As a result, USDC in the program could become stuck.

Recommendations:

It is recommended to require `new_admin_authority` to be the signer in `TransferAdminAuthority`.

Alien: Acknowledged.

[I-4] Incorrect parameter naming

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- set_sale_activity.rs

Description:

The `set_sale_activity()` instruction accepts a boolean parameter to control the enabling and disabling of `is_sale_active`.

```
pub fn apply(ctx: &mut Context<SetSaleActivity>, claim_is_active: bool) →
    Result<()> {
    ctx.accounts.global_config_account.is_sale_active = claim_is_active;
    Ok(())
}
```

However, this variable is incorrectly named `claim_is_active`, which appears to have been copied from another instruction. This could be misleading when reading the code and may hinder maintainability.

Recommendations:

It is recommended to rename the parameter to `sale_is_active`.

Alien: Resolved with `@cdd2d34715 ...`

Zenith: Verified.