// Solana Program Security Assessment 09.25.2024 - 10.08.2024

USDC Pool

Solayer

HALBORN

USDC Pool - Solayer

Prepared by: HALBORN

Last Updated 10/14/2024

Date of Engagement by: September 25th, 2024 - October 8th, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
3	0	0	0	0	3

TABLE OF CONTENTS

1. Introduction
2. Assessment summary
3. Test approach and methodology
4. Risk methodology
5. Scope
6. Assessment summary & findings overview
7. Findings & Tech Details
   - 7.1 Centralization risk
   - 7.2 Use of 'msg!' consumes additional computational budget
   - 7.3 Missing multi-step authority transfer mechanism
8. Automated Testing

1. Introduction

Solayer team engaged Halborn to conduct a security assessment on their USDC Pool **Solana program** beginning on **September 25th, 2024**, and ending on **October, 8th, 2024**. The security assessment was scoped to the Solana Program provided in solayer-labs/usdc-pool-program GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

The USDC Pool program has both administrative and user-facing instructions, and is a system designed to be fully async from users' perspective. Users are provided with a proof of deposit or withdraw (PDA), and the platform handles the withdraw and deposit requests in batch. As Solayer is currently leveraging the OpenEden yield generation protocol, these batch transactions are essentially USDC and TBill token transfers between Solayer and OpenEden ATAs, in order to be able to mint/burn the respective amount of sUSD (Solayer USD).

The batch transactions are scheduled to occur once a day, and are handled by the Off-Chain Operator, which is the signer for crucial operations of the USDC Pool program.

2. Assessment Summary

Halborn was provided **2 weeks** for the engagement and assigned one full-time security engineer to review the security of the Solana Program in scope. The engineer is a blockchain and smart contract security expert with advanced smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the USDC Pool Solana Program.
- Ensure that the program's functionality operates as intended.

In summary, Halborn identified some non-critical issues, that were acknowledged by the Solayer team:

- Centralization Risk.
- Use of msg! consumes additional computational budget.
- Missing two-step authority transfer mechanism.

Overall, the USDC Pool program in-scope is adherent to Solana's best-practices and carries consistent code quality.

3. Test Approach And Methodology

`Halborn` performed a combination of a manual review of the source code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the program assessment. While manual testing is recommended to uncover flaws in business logic, processes, and implementation; automated testing techniques help enhance coverage of programs and can quickly identify items that do not follow security best practices.

The following phases and associated tools were used throughout the term of the assessment:

- Research into the architecture, purpose, and use of the platform.
- Manual program source code review to identify business logic issues.
- Mapping out possible attack vectors.
- Thorough assessment of safety and usage of critical Rust variables and functions in scope that could lead to arithmetic vulnerabilities.
- Scanning dependencies for known vulnerabilities (`cargo audit`).
- Local runtime testing (`solana-test-framework`).

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4

SEVERITY	SCORE VALUE RANGE
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: usdc-pool-program

(b) Assessed Commit ID: d91a5d7

(c) Items in scope:

- ./Cargo.toml
- ./src/contexts/update_openeden_info.rs
- ./src/contexts/update_susd_metadata.rs
- ./src/contexts/initiate_batch_deposit.rs
- ./src/contexts/withdraw.rs
- ./src/contexts/update_susd_rate.rs
- ./src/contexts/initialize.rs
- ./src/contexts/deposit.rs
- ./src/contexts/set_operator.rs
- ./src/contexts/set_fee.rs
- ./src/contexts/resolve_batch_deposit.rs
- ./src/contexts/collect_fee.rs
- ./src/contexts/initiate_batch_withdraw.rs
- ./src/contexts/mod.rs
- ./src/contexts/set_rate_authority.rs
- ./src/contexts/emergency_transfer_rwa_token.rs
- ./src/contexts/resolve_batch_withdraw.rs
- ./src/constants.rs
- ./src/lib.rs
- ./src/state/depositproof.rs
- ./src/state/withdrawproof.rs
- ./src/state/mod.rs
- ./src/state/openeden_info.rs
- ./src/state/pool.rs
- ./src/errors.rs
- ./src/utils.rs

Out-of-Scope:

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	3

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
CENTRALIZATION RISK	INFORMATIONAL	ACKNOWLEDGED - 10/10/2024
USE OF 'MSG!' CONSUMES ADDITIONAL COMPUTATIONAL BUDGET	INFORMATIONAL	ACKNOWLEDGED - 10/10/2024
MISSING MULTI-STEP AUTHORITY TRANSFER MECHANISM	INFORMATIONAL	ACKNOWLEDGED - 10/10/2024

7. FINDINGS & TECH DETAILS

7.1 CENTRALIZATION RISK

// INFORMATIONAL

Description

Currently, the `operator` account is the **only signer** of critical batch deposit and withdraw operations, handling significant amount of tokens multiple tokens, such as `USDC`, `TBill` and `sUSD (Solayer USD)`. It is not mentioned in the documentation that it is a requirement for the `operator` account to be a multi-signature, such as Squads, neither the instructions require more than one signer, in order to leverage granularity.

In a scenario where the `operator` account gets compromised or the access to this account is lost, the entire functionality of the platform is compromised.

- programs/usdc-pool-program/src/contexts/initiate_batch_deposit.rs

```
#[derive(Accounts)]
pub struct InitiateBatchDeposit<'info> {
    #[account(mut)]
    operator: Signer<'info>,
```

- programs/usdc-pool-program/src/contexts/initiate_batch_withdraw.rs

```
#[derive(Accounts)]
pub struct InitiateBatchWithdraw<'info> {
    #[account(mut)]
    operator: Signer<'info>,
```

- programs/usdc-pool-program/src/contexts/resolve_batch_deposit.rs

```
#[derive(Accounts)]
pub struct ResolveBatchDeposit<'info> {
    #[account(mut)]
    operator: Signer<'info>,
```

- programs/usdc-pool-program/src/contexts/resolve_batch_withdraw.rs

```rust
#[derive(Accounts)]
pub struct ResolveBatchWithdraw<'info> {
    #[account(mut)]
    operator: Signer<'info>,
```

- programs/usdc-pool-program/src/contexts/collect_fee.rs

```rust
#[derive(Accounts)]
pub struct CollectFee<'info> {
    #[account(mut)]
    operator: Signer<'info>,
```

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation

It is recommended to add a more granular access-control mechanism to mission-critical functions, which are currently relying on solely in the `Operator` account.
Consider utilizing a Multi-signature wallet for the `Operator` account. Alternatively, require more than a single signature for the mentioned instructions.

Remediation

ACKNOWLEDGED: The **Solayer team** has acknowledged the issue.

7.2 USE OF 'MSG!' CONSUMES ADDITIONAL COMPUTATIONAL BUDGET

// INFORMATIONAL

Description

The usage of `msg!` is usually advisable during tests, and will incur in additional computational budget when the instruction is processed in Mainnet.
In both `resolve_batch_deposit` and `resolve_batch_withdraw` instructions, it was observed logging utilizing `msg!`.

– programs/usdc-pool-program/src/contexts/resolve_batch_deposit.rs

```
            msg!("expected_amount {:?}", expected_amount);
            msg!("rwa_token_amount {:?}", rwa_token_amount);
            msg!("susd_mint_amount {:?}", susd_mint_amount);
```

– programs/usdc-pool-program/src/contexts/resolve_batch_withdraw.rs

```
            msg!("expected_amount {:?}", expected_amount);
            msg!("usdc_withdraw_amount {:?}", usdc_withdraw_amount);
```

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation

Consider removing `msg!` logging before mainnet deployment for enhanced computational budget efficiency.

Remediation

ACKNOWLEDGED: The **Solayer team** has acknowledged the issue.

7.3 MISSING MULTI-STEP AUTHORITY TRANSFER MECHANISM

// INFORMATIONAL

Description

The existing implementation of the `set_operator` and `set_rate_authority` instructions employ an one-step procedure for authority delegation, which presents a security concern. This method lacks a safeguard against inadvertent delegations to undesired accounts.

– programs/usdc-pool-program/src/contexts/set_operator.rs

```
#[derive(Accounts)]
pub struct SetOperator<'info> {
    #[account(mut)]
    manager: Signer<'info>,
    #[account(
        mut,
        constraint = new_operator.key() != pool.operator.key()
    )]
    new_operator: Signer<'info>,
    #[account(
        mut,
        has_one = manager,
        seeds = [b"pool", USDC_MINT.as_ref(), pool.susd_mint.as_ref()],
        bump = pool.bump
    )]
    pool: Account<'info, Pool>,
}

impl<'info> SetOperator<'info> {
    pub fn set_operator(&mut self) -> Result<()> {
        self.pool.operator = self.new_operator.key();
        Ok(())
    }
}
```

– programs/usdc-pool-program/src/contexts/set_rate_authority.rs

```
#[derive(Accounts)]
pub struct SetRateAuthority<'info> {
    #[account(mut)]
```

```
    manager: Signer<'info>,
    #[account(
        mut,
        constraint = new_rate_authority.key() != pool.rate_authority.key()
    )]
    /// CHECK: The new rate authority account
    new_rate_authority: AccountInfo<'info>,
    #[account(
        mut,
        has_one = manager,
        seeds = [b"pool", USDC_MINT.as_ref(), pool.susd_mint.as_ref()],
        bump = pool.bump
    )]
    pool: Account<'info, Pool>,
}

impl<'info> SetRateAuthority<'info> {
    pub fn set_rate_authority(&mut self) -> Result<()> {
        self.pool.rate_authority = self.new_rate_authority.key();
        Ok(())
    }
}
```

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation

To resolve this issue, it is advisable to establish a multi-step process for `authority` transfer (being `rate_authority` or `new_operator`), thereby enhancing the security of the operation. The current `authority` would first propose a new candidate `authority`, who would then need to formally **accept** the role.
This process would be structured as follows:

1. **Proposal by Current Authority:** The current signer proposes a new candidate signer. This action updates the `candidate_authority` field in the account's state. This step assumes the prior creation of an additional field `candidate_authority` the state.
2. **Acceptance by New Authority:** The proposed `candidate_authority` formally **accepts the role**. This step transfers the `authority` status from the current `authority` to the `candidate_authority`.

Remediation

ACKNOWLEDGED: The **Solayer team** has acknowledged the issue.

8. AUTOMATED TESTING

STATIC ANALYSIS REPORT

Description

Halborn used automated security scanners to assist with detection of well-known security issues and vulnerabilities. Among the tools used was `cargo audit`, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in `https://crates.io` are stored in a repository named The RustSec Advisory Database. `cargo audit` is a human-readable version of the advisory database which performs a scanning on Cargo.lock. Security Detections are only in scope. All vulnerabilities shown here were already disclosed in the above report. However, to better assist the developers maintaining this code, the auditors are including the output with the dependencies tree, and this is included in the cargo audit output to better know the dependencies affected by unmaintained and vulnerable crates.

Cargo Audit Results

ID	CRATE	DESCCRIPTION
RUSTSEC-2022-0093	ed25519-dalek	Double Public Key Signing Function Oracle Attack on `ed255109-dalek`
RUSTSEC-2024-0344	curve25519-dalek	Timing variability in `curve25519-dalek`'s Scalar29::sub/Scalar52::sub
RUSTSEC-2021-0145	atty	Potential unaligned read

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.