Mayan

Audit

Presented by:

OtterSec contact@osec.io

Robert Chen notdeghost@osec.io

Harrison Green hgarrereyn@osec.io

Akash Gurugunti Sud0u53r.ak@osec.io

Contents

Appendices

01 | Executive Summary

Overview

Mayan Finance engaged OtterSec to perform an assessment of the `solana-programs` and `swap-bridge` programs. This assessment was conducted between December 2nd, 2022 and January 6th, 2023. For more information on our auditing methodology, see Appendix B.

Critical vulnerabilities were communicated to the team prior to the delivery of the report to speed up remediation. After delivering our audit report, we worked closely with the team to streamline patches and confirm remediation. We delivered final confirmation of the patches January 9th, 2023.

Key Findings

Over the course of this audit engagement, we produced 9 findings total.

In particular, we found a critical issue which could lead to loss of user funds (OS-MYN-ADV-00), as well as some denial of service concerns while performing swaps leading to the locking of user funds (OS-MYN-ADV-01, OS-MYN-ADV-02, OS-MYN-ADV-03).

We also made suggestions around tighter access control around critical signer seeds (OS-MYN-SUG-00) and general refactors (OS-MYN-SUG-01).

Overall, we commend the Mayan Finance team for being responsive and knowledgeable throughout the audit.

02 | Scope

The source code was delivered to us in a git repositories at github.com/mayan-finance/solana-programs and github.com/mayan-finance/swap-bridge. This audit was performed against commits `e323616` and `a46288f` respectively.

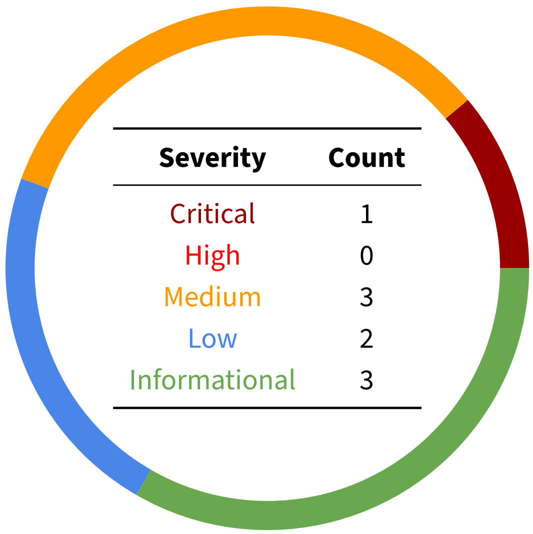
A brief description of the programs is as follows.

Name	Description
solana-programs	Solana implementation for Mayan cross-chain swap auction protocol.
swap-bridge	Contracts for publishing and verifying cross-chain swap messages using Wormhole message passing and token bridge.

03 | Findings

Overall, we reported 9 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings don't have an immediate impact but will help mitigate future vulnerabilities.



Severity	Count
Critical	1
High	0
Medium	3
Low	2
Informational	3

04 | Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-MYN-ADV-00	Critical	Resolved	Relayer can steal tokens while performing flash swap leading to loss of user funds.
OS-MYN-ADV-01	Medium	Resolved	A malicious user can cause DoS for other users while performing their swaps by using different token accounts.
OS-MYN-ADV-02	Medium	Resolved	A malicious user can cause DoS for other users while performing their swaps by using claim instruction with unintended inputs.
OS-MYN-ADV-03	Medium	Resolved	A malicious user can cause DoS for other users while performing their swaps by initializing state before claim using init from solana instruction.
OS-MYN-ADV-04	Low	Resolved	Extra fee is collected from payer and sent to token bridge fee account unnecessarily.
OS-MYN-ADV-05	Low	Resolved	Dust amount while performing swaps is left in the contract.

OS-MYN-ADV-00 [crit] [resolved] | Stealing Tokens While Swapping

Description

In `mayan_flash_swap_finish` instruction, the `spl_transfer` function transfers the tokens from `from_acc` to the `to_acc` to complete the swap. Here, the `spl_transfer` function uses the `mayan_invoke` function internally, which makes all of the Cross-Program Invocations(CPI) with `sol_invoke_signed`, since the `ctx->invoke_with_seed` is set to true on the initialization of the context of an instruction.

Now, a malicious user can perform the swap, and while calling the `mayan_flash_swap_finish` instruction, they can pass in the token account of the main PDA account, for which bridged tokens are transferred (like ATA) and finish the swap. This performs the transfer between token accounts of main account, essentially giving free tokens to the malicious user.

Proof of Concept

1. User1 calls the swap function on the ethereum contract with ATA of the main PDA account as `recepient.mayanAddr` to swap X tokens for Y, which transfers the funds to that address.
2. Another user User2 (malicious) calls the swap function on the ethereum contract to swap Y tokens to X.
3. Now, a malicious user can call the `mayan_flash_swap_start` instruction to collect the from tokens and call `mayan_flash_swap_finish` instruction with ATA of main PDA account on X mint as `from_acc` and `to_acc`, which transfers the tokens from and to the same account.
4. The swap is considered as completed, but the tokens that should be transferred to the `to_acc` at the end of the swap are not transferred to it.

Remediation

This can be remediated by separating the `mayan_invoke` into two functions, `mayan_invoke` and `mayan_invoke_signed` and using signed invocation only when necessary.

Patch

Fixed by separating the CPI `mayan_invoke` function into two separate functions with and without seeds in efae9ee.

OS-MYN-ADV-01 [med] [resolved] | DOS While Performing Swap - 1

Description

In `mayan_flash_swap_start` instruction, the `validate_flash_swap_start` function validates the accounts passed to that instruction. It does not validate whether the `from_acc` passed in was in fact the account that received tokens for performing the current swap.

A malicious user can bridge funds to a different token account under the main account's ownership and use another token account of the main that is used by another genuine user to complete the flash swap. This results in denial of service for the genuine user when trying to do the swap.

Proof of Concept

1. User1 calls the `swap` function on the ethereum contract with ATA of the main PDA account as `recepient.mayanAddr`, which transfers the funds to that address
2. Another user, User2 (malicious), calls the `swap` function on the ethereum contract with some token account of the main PDA account (not ATA) as `recepient.mayanAddr`
3. Now, before the User1 can perform the swap, User2 claims using `mayan_claim` instruction and performs swap with ATA of the main PDA account as `from_acc`.
4. The swap is completed by the User2. Now, if User1 tries to perform the swap with ATA of the main PDA account, the swap fails, as the tokens are already used by User2 to perform their swap.

Remediation

This can be remediate by storing the `to_acc` on mayan state and validating it while doing the swap or by asserting then token account used to be Associated Token Account(ATA) of the main PDA account while bridging the funds and doing the swap.

Patch

Fixed by asserting `msg1.target_addr` to be ATA of main PDA account in efae9ee.

OS-MYN-ADV-02 [med] [resolved] | DOS While Performing Swap - 2

Description

In `claim` instruction, the `validate_mint_accounts` function validates the `mint_from` and `mint_to` accounts passed in with the PDAs generated with given token address, chain id and nonce.

A malicious user can call the `claim` instruction with non-canonical `mint_to_nonce` and its relevant PDA. Then the instruction stores that `mint_to` as the destination token address. This makes the state unable to swap, since the `mint_to` is not a valid token mint address. Although the user will be able to retrieve his funds after the specified deadline is over, a malicious user can perform this multiple times on all the incoming swaps and make the protocol unusable for the genuine users.

Proof of Concept

1. A user calls the `swap` function on the ethereum contract to start the swap.
2. A malicious user then calls the `claim` instruction with non-canonical `mint_to_nonce` and its relevant PDA.
3. Now, if a user/relayer tries to perform swap on the state using the `mayan_flash_swap_start` instruction, the user has to transfer `mint_to` tokens in the `mayan_flash_swap_finish` instruction, which is not possible since a token mint account doesn't exist at that address.
4. Now the state is rendered useless and the funds can be retrieved by the user only after the specified deadline is over.

Remediation

This can be remediated by calculating the PDA of the `mint_from` and `mint_to` addresses on-chain with canonical bumps instead of taking them from the user input.

Patch

Resolved in 4539a1d.

OS-MYN-ADV-03 [med] [resolved] | DOS While Performing Swap - 3

Description

In `init_from_solana` instruction, a state PDA account is created with given `msg1` and `msg2` accounts. Then, `mint_from` tokens are collected from the user to main ATA and a state is created with the given swap details.

A malicious user can pass in `msg1` and `msg2` accounts of a genuine swap to the `init_from_solana` instruction and create initialize a state on that before `claim` instruction is called on them. This makes the program throw when trying to create swap state for the genuine swap using the `claim` instruction.

Proof of Concept

1. A user calls the `swap` function on the ethereum contract to start the swap.
2. A malicious user then calls the `init_from_solana` instruction with genuine `msg1` and `msg2` addresses and initializes a state on that account.
3. Now, if the genuine user/relayer tries to initialize state on those msg accounts using the `claim` instruction, the program throws error since the state is already initialized at the address.

Remediation

Since the `msg1` and `msg2` accounts are expected to be random addresses, this can be remediated by asserting the msg accounts to be signers.

Patch

Resolved in 4539a1d.

OS-MYN-ADV-04 [low] [resolved] | Unnecessary Extra Fee Consumption

Description

In `mayan_trx` instruction, the fee for bridging tokens is taken as input from data where that amount is transferred from the payer account to fee account of the token bridge

This is redundant as the token bridge already has implementation that takes the fee amount required to bridge the tokens directly from the payer (here)

flash-swap/src/mayanswap.c C

```c
    result = system_transfer(ctx, trx.owner->key,
     ↪  trx.transfer.fee_acc->key, trx.transfer.fee);
    if (result != SUCCESS)
    {
        mayan_error("cannot transfer fee");
        return result;
    }
```

Remediation

The unnecessary `system_transfer` function that transfers fee from payer account to fee account should be removed.

Patch

Unnecessary `system_transfer` function call is removed in efae9ee.

OS-MYN-ADV-05 [low] [resolved] | Dust Amount Stuck In Contract

Description

The `swap` function in the ethereum contract is called by the user to swap tokens through wormhole. The `amountIn` number of tokens are transferred from the user account to the contract account, then are transferred through the token bridge. However, the token bridge performs `normalize` and `denormalize` on the amount to be transferred in order to remove the dust amount. So, if the user transfers tokens with dust amount, they will get stuck in the contract and cannot be retrieved.

Similarly, the `wrapAndSwapETH` function also uses the token bridge which returns the dust amount of ETH back to the contract, which cannot be retrieved and becomes stuck.

Proof of Concept

1. Call the `swap` function to swap tokens that have more than 8 decimals and an `amountIn` value with no trailing zeroes.
2. You will see that only the `denormalize(normalize(amountIn))` value is transferred by the token bridge and the remaining amount is left in the contract balance.

Remediation

This can be remediated by either taking only the required amount of tokens from the user (without dust amounts) or by having a functionality for the guardian to collect the dust amounts left in the contract.

Patch

Resolved in #1.

05 | General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent antipatterns and could lead to security issues in the future.

ID	Description
OS-MYN-SUG-00	Unnecessary invocation of all cross-program invocations with signer seeds of all PDAs.
OS-MYN-SUG-01	Consider using more appropriate and declarative names for functions
OS-MYN-SUG-02	Code refactoring by removing unnecessary function parameters, struct fields, etc.

OS-MYN-SUG-00 [resolved] | Unnecessary Signed Cross Program Invocation

Description

The `mayan_invoke` function checks for a context variable `ctx->invoke_with_seed` and based on that, it makes Cross-Program Invocation(CPI) with or without seeds. Since the `ctx->invoke_with_seed` is set to true in `ctx_init` function, which is called from entrypoint, every instruction has this value set to true and all CPIs are invoked with signer seeds of all PDAs.

This can cause unintended behaviors where a CPI is expected to be signed by the instruction caller, but instead, all the PDA accounts are passed in as signers to the CPI.

flash-swap/src/ctx.c C

```c
u64 ctx_init(struct prog_ctx *ctx, SolParameters *params)
{
        ctx->params = params;
        ctx->prog_id = params->program_id;

        ctx->invoke_with_seed = true;
```

flash-swap/src/ctx.h C

```c
inline static u64 mayan_invoke(const struct prog_ctx *ctx,
                               const SolInstruction *ix)
{
    if (ctx->invoke_with_seed) {
        return sol_invoke_signed(ix, ctx->params->ka,
                     ctx->params->ka_num, ctx->seeds,
                     SOL_ARRAY_SIZE(ctx->seeds));
    }
    return sol_invoke(ix, ctx->params->ka, ctx->params->ka_num);
}
```

Remediation

This can be remediated by defining two separate functions for CPI, `mayan_invoke` that does CPI without seeds and `mayan_invoke_signed` that does the CPI with signer seeds of PDAs.

Patch

Resolved in efae9ee.

OS-MYN-SUG-01 [resolved] | More Appropriate Function Names

Description

The function `ctx_check_state_addr` checks if the given state account is equal to the Program Derived Address(PDA) generated with state seeds stored on the ctx.

Remediation

A more appropriate name for `ctx_check_state_addr` function would be `ctx_check_seed_addr`.

Description

The function `mayan_data_rate` returns the rate for the swap before the swap and returns the sequence ID of the Wormhole VAA message after the swap.

Remediation

If the rate and sequence ID are not required, then it is recommended to remove the field from the state. Otherwise, a more appropriate name for the `mayan_data_rate` function would be `mayan_data_seq_id`, since the rate is an unused field.

Patch

Resolved in efae9ee.

OS-MYN-SUG-02 [resolved] | Code Refactoring

Description

The `write_*` functions in `utils.h` are used to write given `data` to a given `data_ptr` and increment the `data_ptr` by the size of the data written to it. But, the `data` parameter passed into them are not used anywhere in the function.

Remediation

It is recommended to remove the unnecessary data parameter in the `write_*` functions.

Description

The `swap_delegate_seed` field and its related functions are not used anywhere in the program. The `decimal` field in state PDA account is also not used anywhere in the program. There are some `SolPubkeys` hardcoded in the program like `SWAP_PROGRAM_ID`, `SWIM_6_PROGRAM_ID`, etc.

Remediation

It is recommended to remove the unnecessary parts of the code mentioned above.

Description

In `transfer_inchain_spl` function, the `state.state` is set to `STATE_DONE_SWAPPED` but this is an improper state and is also unnecessary, since the state is overwritten with proper value again in the `transfer_inchain` function. And in the `wh_check_claimed` function, it returns false when `chain_id` doesn't match with any given chains. Since the return value of the function is `u64`, the false value is considered as `0`, which is equal to `SUCCESS`. Therefore, in the error case, the function returns `SUCCESS`.

Remediation

It is recommended to remove the unnecessary assignment of state in `transfer_inchain_spl` function. Also, the `ERROR_CUSTOM_ZERO` should be returned instead of returning false when `chain_id` doesn't match any of the given chains.

Patch

Resolved in efae9ee and e323616.

A | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the General Findings section.

Critical	Vulnerabilities that immediately lead to loss of user funds with minimal preconditions Examples: • Misconfigured authority or access control validation • Improperly designed economic incentives leading to loss of funds
High	Vulnerabilities that could lead to loss of user funds but are potentially difficult to exploit. Examples: • Loss of funds requiring specific victim interactions • Exploitation involving high capital requirement with respect to payout
Medium	Vulnerabilities that could lead to denial of service scenarios or degraded usability. Examples: • Malicious input that causes computational limit exhaustion • Forced exceptions in normal user flow
Low	Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions
Informational	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants • Improved input validation

B | Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the implementation of the program requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of sum, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to get a comprehensive understanding of the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.