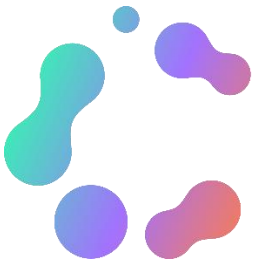
Code Audit

for

Coral



CORAL

Commissioned by

Streamflow Finance

Streamflow FINANCE

https://opcodes.fr / contact@opcodes.fr

15 Rue des Halles, 75001 Paris, France

Project Information

Project	
Mission	Code audit
Client	Streamflow Finance
Start Date	08/25/2022
End Date	08/29/2022

Document Revision			
Version	Date	Details	Authors
1.0	08/30/2022	Document creation	Thibault MARBOUD Xavier BRUNI
1.1	08/31/2022	Peer review	Baptiste OUERIAGLI

Table of contents

Overview

Mission Context

The purpose of the mission was to perform a code audit to discover issues and vulnerabilities in the mission scope. Comprehensive testing has been performed using automated and manual testing techniques.

Mission Scope

As defined with Streamflow Finance before the mission, the scope of this assessment was a multisignature Solana program made by Coral. The source code is open source and available on the following GitHub repository:

- https://github.com/coral-xyz/multisig / a413b76 (main)

OPCODES engineers were due to strictly respect the perimeter agreed with Streamflow Finance as well as respect ethical hacking behavior.

Project Summary

Streamflow Finance is building a token vesting application on Solana and wants to support multisignature.

Multisignature (or multisig) is a solution requiring multiple parties to agree before sending a transaction. OPCODES engineers encourage Streamflow Finance to use multisig as it will add an additional security layer to their vesting application. Multisig makes even more sense considering the recent events with Slope' wallets hack.

Coral multisig program leverage anchor framework and is relatively short with 400 lines of code. The program was deployed on 9 different addresses from which 8500 transactions were processed over a total of 1168 multisig.

Synthesis

Security Level: GOOD

The overall security level is considered as good. Coral multisig program correctly implements a multisignature system and can be safely used by Streamflow Finance.

The assessment demonstrated the presence of 1 medium vulnerability. The multisig program does not check if a target program has been updated between the transaction creation and its execution. Multisig parties could approve on a transaction that would result in a different outcome that what they were expecting.

Three minor vulnerabilities have also been reported. They concern a low probability integer overflow, an arbitrary nonce that could lead to an unusable multisig and logic bug regarding threshold update.

OPCODES also added 3 informational issues to this report. They represent possible improvements and do not lead to any exploitable scenario but may enforce bad practices.

Vulnerabilities summary

Total vulnerabilities	7
Critical	0
Major	0
Medium	1
Minor	3
Informational	3

Vulnerabilities & issues table

Identified vulnerabilities

Ref	Vulnerability title	Severity	Remediation effort
#1	Missing check against an approved program upgrade	Medium	Medium
#2	Arbitrary nonce can lead to unusable multisig	Minor	Low
#3	Integer overflow on sequence number	Minor	Low
#4	Threshold update does not increment sequence number	Minor	Low
#5	Users can't deny transaction	Informative	Medium
#6	Code optimization	Informative	Medium
#7	Outdated dependencies	Informative	Low

Identified vulnerabilities

Missing check against an approved program upgrade

Severity	Remediation effort
Medium	Medium

Description

Solana programs can be upgraded, and an update could occur between a transaction creation and its execution.

OPCODES engineers think that this issue is relevant as a lot of Solana programs are being upgraded in the shadows, without users knowing it.

Scope

Multisig program

Risk

A targeted program could be upgraded between a transaction creation/approval and its execution. This could result in an unintended behavior that the involved parties did not originally agree on.

Remediation

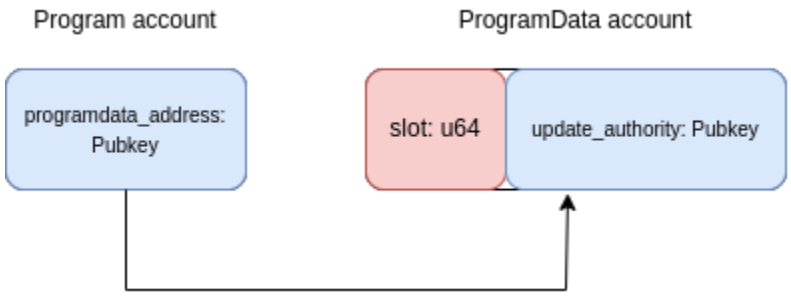
OPCODES engineers would recommend adding a security check in the `execute_transaction` instruction to ensure that the target program was not updated.

This can be done by first, storing the slot upon a transaction creation.

```
  pub fn create_transaction(
      ctx: Context<CreateTransaction>,
      pid: Pubkey,
      accs: Vec<TransactionAccount>,
      data: Vec<u8>,
  ) -> Result<()> {
    [...]
    let tx = &mut ctx.accounts.transaction;
    [...]
+   let clock = Clock::get()?;
+   tx.slot = clock.slot - 1;
    Ok(())
  }
```

```
  #[account]
  pub struct Transaction {
    [...]
    pub did_execute: bool,
    // Owner set sequence number.
    pub owner_set_seqno: u32,
+   // Transaction creation slot
+   pub slot: u64
  }
```

Then, when the transaction is about to be executed, the multisig program should read the *ProgramData* account and ensure that the slot is not bigger than the transaction's slot. The multisig program should also ensure that the *ProgramData* account is legit by deserializing the *Program* account and comparing both public keys.



Note: Native programs don't have a `ProgramData` account, this is the case for the System program and BPFLoader program. A whitelist may be needed to skip this security check for those programs.

Note: Adding a new field in the `Transaction` struct will result in a more expensive rent. Frontend applications may have to update their code to ensure that they are creating rent-exempted accounts.

Arbitrary nonce can lead to unusable multisig

Severity	Remediation effort
Minor	Low

Description

Upon the creation of a new multisig account, users must pass the nonce of the *multisig_signer* account that will be the PDA used to sign the transactions.

programs/multisig/src/lib.rs (L33)

```
pub fn create_multisig(
    ctx: Context<CreateMultisig>,
    owners: Vec<Pubkey>,
    threshold: u64,
    nonce: u8,
) -> Result<()> {
    assert_unique_owners(&owners)?;
    require!(
        threshold > 0 && threshold <= owners.len() as u64,
        InvalidThreshold
    );
    require!(!owners.is_empty(), InvalidOwnersLen);

    let multisig = &mut ctx.accounts.multisig;
    multisig.owners = owners;
    multisig.threshold = threshold;
    multisig.nonce = nonce;
    multisig.owner_set_seqno = 0;
    Ok(())
}
```

Scope

Multisig program

Risk

It is possible to create a *multisig* account with an arbitrary nonce that will lead to a *multisig_signer* account that does lie on the ed25519 curve. This would result in a multisig

that is unable to sign transactions. It could also be used as a social attack in order to trick parties into giving authority to a broken multisig.

Remediation

OPCODES engineers recommend adding a security check in the *`create_multisig`* instruction to ensure that the given nonce is correct.

It can be done by with the *`create_program_address`* method of the Pubkey struct:

https://docs.rs/solana-program/1.11.10/solana_program/pubkey/struct.Pubkey.html

Integer overflow on sequence number

Severity	Remediation effort
Minor	Low

Description

When updating the owners of a multisig, the program unsafely increments a sequence number.

programs/multisig/src/lib.rs (L121)

```
pub fn set_owners(ctx: Context<Auth>, owners: Vec<Pubkey>) -> Result<()> {
    assert_unique_owners(&owners)?;
    require!(!owners.is_empty(), InvalidOwnersLen);

    let multisig = &mut ctx.accounts.multisig;

    if (owners.len() as u64) < multisig.threshold {
        multisig.threshold = owners.len() as u64;
    }

    multisig.owners = owners;
    multisig.owner_set_seqno += 1;

    Ok(())
}
```

Scope

Multisig program

Risk

The field *owner_set_seqno* could overflow and previous transactions with different owners could be approved and executed.

Remediation

Even if the probability is low, OPCODES engineers recommend adding an overflow check. It can be globally by adding the following lines to the *cargo.toml* file.

```
[profile.release]
overflow-checks = true
```

Alternatively, when it makes more sense to do overflow checks on case-by-case basis the following functions should be used: *checked_mul*, *checked_div*, *checked_add* or *checked_sub*.

```
multisig.owner_set_seqno = multisig
    .owner_set_seqno
    .checked_add(1)
    .ok_or(ErrorCode::Overflow)?;
```

Note: There already is an unused error code for overflows.

Threshold update does not increment sequence number

Severity	Remediation effort
Minor	Low

Description

Coral Multisig program allows parties to change the threshold number, which represent the number of users required to sign a transaction before it can be executed.

Updating the threshold does not invalidate previously created transaction.

Scope

Multisig program

Risk

Multisig parties could accept to reduce the threshold number thinking that transactions created before will still need the previous threshold.

Remediation

As a precaution, OPCODES engineers recommend incrementing the `owner_set_seqno` inside the `change_threshold` instruction. It will prevent the execution of transactions created before a threshold update.

Users can't deny transaction

Severity	Remediation effort
Informational	Medium

Description

A transaction cannot be refused by parties.

Scope

Multisig program

Risk

This is an informational issue; It does not lead to any exploitable scenario. But over time transactions may accumulate and a user could inadvertently accept the wrong transaction.

Remediation

OPCODES engineers think that it makes sense to have a *deny_transaction* allowing users to refuse the signature of a transaction.

Code optimization

Severity	Remediation effort
Informational	Medium

Description

Some part of the code could be simplified and optimized, leading to a more straightforward and easier to maintain code.

Scope

Multisig program

Risk

This is an informational issue; It does not lead to any exploitable scenario.

Remediation

OPCODES engineers would recommend adding a check in the *create_transaction* to ensure that the *multisig_signer* account is indeed present as a signer. If not, the transaction should be refused, as it does not require the multisig signature.

After adding this check, the *execute_transaction* instruction could be simplified. Indeed, the following lines could be deleted.

programs/multisig/src/lib.rs (L171)

```
  let mut ix: Instruction = (*ctx.accounts.transaction).deref().into();
- ix.accounts = ix
-     .accounts
-     .iter()
-     .map(|acc| {
-         let mut acc = acc.clone();
-         if &acc.pubkey == ctx.accounts.multisig_signer.key {
-             acc.is_signer = true;
-         }
```

```
-         acc
-     })
-     .collect();
  let multisig_key = ctx.accounts.multisig.key();
  let seeds = &[multisig_key.as_ref(), &[ctx.accounts.multisig.nonce]];
  let signer = &[&seeds[..]];
  let accounts = ctx.remaining_accounts;
  solana_program::program::invoke_signed(&ix, accounts, signer)?;
```

Now, the *multisig_signer* account is unnecessary, and it can be removed from the *ExecuteTransaction* struct, reducing the transaction size.

programs/multisig/src/lib.rs (L231)

```
  #[derive(Accounts)]
  pub struct ExecuteTransaction<'info> {
     #[account(constraint = multisig.owner_set_seqno == transaction.owner_set_seqno)]
     multisig: Box<Account<'info, Multisig>>,
-    /// CHECK: multisig_signer is a PDA program signer. Data is never read or …
-    #[account(
-        seeds = [multisig.key().as_ref()],
-        bump = multisig.nonce,
-    )]
-    multisig_signer: UncheckedAccount<'info>,
     #[account(mut, has_one = multisig)]
     transaction: Box<Account<'info, Transaction>>,
  }
```

Outdated dependencies

Severity	Remediation effort
Informative	Low

Description

The following crate could be updated:

Crate	Current version	Latest version
anchor-lang	0.24.2	0.25.0

Scope

Multisig program

Risk

This is an informational issue; At the time of writing this report, it does not represent any risk but may enforce bad practices.

Remediation

It is considered a good practice to update dependencies when possible.

Conclusion

Coral multisig program is a great piece of software that correctly implements multisignatures. The assessment did not report any critical findings. Therefore, the program can be safely used by Streamflow Finance.

OPCODES engineers still recommend fixing the vulnerabilities reported. Especially the medium severity issue regarding the missing program upgrade check before a transaction execution. It will prevent multisig parties to agree on a transaction that would lead to a different outcome that what they were expecting. Concerning the threshold update, OPCODES team think that it should invalidate previously created transaction. Fixing this issue does not require a huge change and will protect unaware users. As a precaution overflow checks should also be implemented.

Finally, informational issues represent possible improvement that could be made to the original program. OPCODES engineers would recommend fixing them if they are meaningful to you.