Franklin Templeton Benji Contracts

Security Assessment (Summary Report)

February 7, 2025

Prepared for:
Franklin Templeton Digital Assets

Prepared by: **Samuel Moelius, Josselin Feist, and Coriolan Pinhas**

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Sam Greenup, Project Manager
sam.greenup@trailofbits.com

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultants were associated with this project:

Samuel Moelius, Consultant
samuel.moelius@trailofbits.com

Coriolan Pinhas, Consultant
coriolan.pinhas@trailofbits.com

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
January 17, 2025	Pre-project kickoff call
January 17, 2025	Delivery of report draft
January 28, 2025	Report readout meeting
February 7, 2024	Delivery of final summary report

Project Targets

The engagement involved a review and testing of the targets listed below.

solana_mutisig

Repository	n/a (source code provided in Zip file)
Version	January 15, 2025
Type	Rust/Anchor
Platform	Solana

solana_mutichain

Repository	n/a (source code provided in Zip file)
Version	January 15, 2025 January 21, 2025 January 22, 2025 January 23, 2025 (one-line update to fix a test)
Type	Rust/Anchor
Platform	Solana

Executive Summary

Engagement Overview

Franklin Templeton engaged Trail of Bits to review the security of its Benji contracts. There are two Benji contracts: a multisignature wallet (`solana_multisig`) and a money market fund (`solana_multichain`). Privileged operations on `solana_multichain` are meant to be called only by `solana_multisig`.

A team of three consultants conducted the review from January 21 to January 28, 2025, for a total of two engineer-weeks of effort. With full access to source code and documentation, we performed static and dynamic testing of the codebase, using automated and manual processes.

Observations and Impact

Our review identified several high-severity issues that could allow a malicious actor to cancel any pending transactions (TOB-FTSOLANA-1), execute malicious transactions (TOB-FTSOLANA-2), and lead to various denial-of-service or spamming attacks (TOB-FTSOLANA-3, TOB-FTSOLANA-5, TOB-FTSOLANA-6). Additionally, we found that several of the programs' functionalities and checks do not work as expected, even without a malicious actor.

Moreover, the codebase significantly lacks documentation and tests; we found that 8 out of the multichain's 23 instructions are not tested. More robust documentation and testing would have made it easier to identify several findings in this report.

Our recommendations from a previous similar code review of the Aptos codebase identified the same shortcomings in testing and the need for better software development practices.

Given the state of the codebase and the nature of the issues found, it is likely that more issues are present, or that future issues will be introduced when the code is updated.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that Franklin Templeton take the following steps:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or any refactor that may occur when addressing other recommendations.

- **Test every instruction before deploying the codebase.** To the extent possible, test every line of every function that executes on chain. This will reduce the likelihood of unexpected failures. The tests should also include negative testing.

- **Prepare documentation.** Create documentation describing the project's two programs (`solana_multisig` and `solana_multichain`), the actors expected to interact with the programs, the actions they can perform, and the accounts the programs maintain. Having such documentation will assist code auditors and reduce the likelihood that the programs will be misused.

- **Identify and document which operations require events for monitoring and create an incident response plan accordingly.** Having a well-defined event strategy will ease the integration of monitoring solutions and will help verify the program's correct behavior.

- **Adopt mature software development processes.** This will help to prevent bugs from entering the codebase. This can include the following approaches:
    - Store your code in a version control system, if you do not already.
    - Establish a well-defined branching strategy.
    - Integrate the tests and linters in the CI.
    - Adopt and enforce a consistent coding style.
    - Enforce minimal unit test coverage.
    - Schedule regular refactoring and document update sessions.

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	While the system benefits from low arithmetic complexity, the code features many uses of `checked_add`, `checked_mul` and `checked_div` that are immediately followed by calls to `unwrap`. Such a pattern defeats the purpose of using checked arithmetic. Saturating arithmetic is used in many places with no explicit justification.	**Moderate**
Auditing	The code features very little logging. It also features many commented-out `msg!` calls, which suggests that logging is used more for debugging than for monitoring the programs' normal operation.	**Weak**
Authentication / Access Controls	The system suffers from a lack of thorough documentation and unclear expectations of the different actors. The `solana_multichain` program contains many missing or incorrect access controls checks.	**Weak**
Complexity Management	The project features large amounts of duplicated code and commented-out code. The commented-out code is difficult to distinguish from intentional comments.	**Weak**
Decentralization	The project is openly centralized.	**Not Applicable**
Documentation	Aside from code comments, the project has no documentation.	**Missing**
Low-Level Manipulation	The project does not use assembly or unsafe Rust. The project does little bit- or byte-level manipulation.	**Not Applicable**

Testing and Verification	At the start of the review, the project's tests could not be run locally. An initial update allowed the tests to run, though not all of them passed. A second update caused all of the tests to pass. The `solana_multichain` program has 23 public functions. Eight of them are not tested at all.	**Weak**
Transaction Ordering	The codebase has no inbuilt slippage protection. However, Franklin Templeton stated that this is by design. The intent is to prevent users from taking advantage of market timing. Additionally, the usage of nonce for shareholders was an undocumented feature that led to a high-severity issue (TOB-FTSOLANA-3). The feature was initially presented as a replay protection mechanism, but further discussion indicates that it is used for transaction ordering; however, its requirements are unclear.	**Not Applicable**

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity	Fix status
1	cancel_self_service_request can be called on any pending transaction	Access Controls	High	Resolved
2	Closing and reopening pending transactions allows a user to execute malicious actions	Data Validation	High	Resolved
3	Shareholders can escape being frozen by increasing their nonce	Data Validation	High	Resolved
4	Incorrect logging of transferred shares	Undefined Behavior	Low	Resolved
5	remove_submitters can remove all submitters	Data Validation	High	Resolved
6	is_frozen is checked on the wrong variable during the transfer of shares	Data Validation	Medium	Resolved
7	Self-service functions can be called when the self-service is disabled	Data Validation	Low	Resolved
8	Frozen account can still cancel transactions	Data Validation	Low	Resolved
9	Code duplication between recover_account and recover_asset	Undefined Behavior	Informational	Unresolved
10	Missing call to is_valid_submitter in AddSubmitters	Data Validation	Informational	Resolved
11	Bump seeds are not stored in PDAs	Denial of Service	Informational	Resolved

12	API key exposure in configuration files	Data Exposure	**Medium**	**Resolved**
13	solana_multisig uses an outdated version of anchor-lang	Patching	**Informational**	**Resolved**
14	Multiple tautologies make the checks always return true	Data Validation	**Informational**	**Resolved**
15	No evidence of linter usage	Patching	**Informational**	**Resolved**
16	Mix of debugging and production code	Patching	**Informational**	**Resolved**
17	Lack of documentation	Patching	**Informational**	**Resolved**
18	Insufficient test coverage	Testing	**Informational**	**Resolved**
19	Insufficient logging	Error Reporting	**Informational**	**Resolved**
20	Incorrect fix pushed mid-review	Patching	**Informational**	**Resolved**

Detailed Findings

1. cancel_self_service_request can be called on any pending transaction	
Severity: **High**	Difficulty: **Medium**
Type: Access Controls	Finding ID: TOB-FTSOLANA-1
Target: `solana-multisig/programs/solana-multichain/src/lib.rs`	

Description

A lack of access controls on `CancelSelfServiceRequest` allows any shareholder to cancel any pending transaction.

Every pending transaction has a `shareholder` field, which is the shareholder's account who created the transaction:

```
#[account]
pub struct TransactionPending {
    // pda account address of the shareholder ( amount of txs and is frozen etc)
    pub shareholder: Pubkey,
    [..]
```

Figure 1.1: solana-multichain/src/lib.rs#L1770-L1773

When a shareholder cancels a given transaction, there is no check ensuring that they created the transaction to be canceled:

```
#[account(mut, has_one = money_market_fund @ ErrorCode::InvalidShareHolder,
constraint = shareholder.nonce == nonce @ ErrorCode::InvalidNonce)]
pub shareholder: Account<'info, ShareHolder>,
#[account(mut, close = authority, constraint = request_id ==
transaction_pending.key())]
pub transaction_pending: Account<'info, TransactionPending>,
```

Figure 1.2: solana-multichain/src/lib.rs#L1452-L1455

As a result, any shareholder can cancel any transaction.

The same issue exists in `CancelRequest`.

Exploit Scenario

Bob creates a pending transaction, which Eve cancels. Bob tries again and creates new transactions, but Eve continuously cancels them.

Recommendations

Short term, check that `transaction_pending.shareholder` is equal to `shareholder` in `CancelSelfServiceRequest` and `CancelRequest`.

Long term, create a diagram showing the life cycle of pending transactions. Create tests highlighting every expected state transition, and transitions that should not happen.

2. Closing and reopening pending transactions allows a user to execute malicious actions

Severity: High	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-FTSOLANA-2
Target: `solana-multisig/programs/solana-multichain/src/lib.rs`	

Description
The lack of validation on the content of transactions allows an attacker to settle invalid transactions.

The normal transaction flow is the following:

- A transaction is requested by a user, and an associated `TransactionPending` is created and initialized by the multichain program:

```
pub struct SetUpAIP<'info> {
    [..]
    #[account(init, payer = authority, space = TransactionPending::LEN)]
    pub transaction_pending: Account<'info, TransactionPending>,
```

Figure 2.1: Example of `TransactionPending` initialization (`lib.rs#L1241-L1249`)

- The transaction is validated through the multisig.
- The `TransactionPending` is then executed through `settle_transaction`.

Before the transaction is validated, the user has the option to cancel a transaction, through `cancel_request` or `cancel_self_service_request`. Canceling a pending transaction closes the associated account:

```
pub struct CancelSelfServiceRequest<'info> {
    [..]
    #[account(mut, close = authority, constraint = request_id ==
transaction_pending.key(), constraint = transaction_pending.shareholder ==
shareholder.key())]
    pub transaction_pending: Account<'info, TransactionPending>,
```

Figure 2.2: Canceling and closing a pending transaction (`lib.rs#L1521-L1529`)

Once an account has been closed, it can be initialized again. `SettleTransaction` checks only for the transaction key, and not its field:

```
pub struct SettleTransaction<'info> {
    [..]
    #[account(mut, close = authority, has_one = shareholder @
ErrorCode::InvalidTransaction, constraint = tx_id == transaction_pending.key())]
    pub transaction_pending: Account<'info, TransactionPending>,
```

Figure 2.1: Settling a transaction (`lib.rs#L1198-L1208`)

As a result, an attacker can initiate a transaction, convince the multisig parties to execute it, and change the transaction's content by closing and reopening it. This allows the attacker to execute any type of transaction as long as one transaction is validated.

Exploit Scenario

- Eve generates a keypair for a pending transaction.
- Eve calls `request_self_service_cash_purchase`, which initializes the transaction's account. The amount is set to 1.
- The multisig validates the transaction and gathers the signatures.
- Before the multisig's transaction is executed, Eve calls `cancel_self_service_request`, which closes the transaction's account.
- She again calls `request_self_service_cash_purchase` with the same transaction's account, but with an amount equal to one million.
- The multisig transaction is executed, and the pending transaction passes with the amount of one million.

Recommendations

Short term, hash the pending transaction field, and compare the expected hash with the current transaction's hash when settling a transaction.

Long term, create a diagram showing the life cycle of pending transactions. Create tests highlighting every expected state transition, and transitions that should not happen.

3. Shareholders can escape being frozen by increasing their nonce	
Severity: High	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-FTSOLANA-3
Target: `solana-multichain/programs/solana-multichain/src/lib.rs`	

Description

The nonce's system on the shareholders allows a malicious shareholder to escape being frozen.

When freezing an account, the nonce's parameter must match the shareholder's current nonce (`shareholder.nonce == nonce`):

```
#[account(mut, has_one = money_market_fund @ ErrorCode::InvalidShareHolder,
constraint = shareholder.nonce == nonce @ ErrorCode::InvalidNonce)]
```

Figure 3.1: `solana-multichain/src/lib.rs#L1619`

However, a shareholder can increase their nonce by making "self-service" calls. If a shareholder knows they are going to be frozen, they can spam the network to prevent this from happening.

Recommendations

Short term, remove the nonce validation on `freeze_shareholder`. Consider whether or not the nonce should be increased in the freeze instruction, and remove it if it should not.

Long term, document the behavior of the shareholder's nonce. The lack of documentation on the nonce and the developers' intent for its use within the code increases the risks of related issues and is likely to introduce issues when the code is updated.

4. Incorrect logging of transferred shares

Severity: **Low**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-FTSOLANA-4
Target: `solana-multisig/programs/solana-multichain/src/lib.rs`	

Description

A type confusion error causes the `request_share_transfer` function's `shares` argument to be passed as the `calculate_shares` function's `amount` argument.

The relevant code appears in figures 4.1 through 4.4. The `request_share_transfer` function takes a `shares` argument that is passed to the `create_pending_transaction` function (figure 4.1). The `create_pending_transaction` function uses the value to initialize the `amount` field of a `TransactionPending` account (figure 4.2). When the `process_settlement` function is called, the `amount` field is passed to `calculate_shares` as an `amount` (figure 4.3). The calculated number of shares is then logged (figure 4.4).

```
pub fn request_share_transfer(
    ctx: Context<RequestShareTransfer>,
    shares: u64,
    trade_date: i64,
    _nonce: u64,
) -> Result<()> {
    // ...
    create_pending_transaction(
        transaction_pending,
        money_market_fund,
        shareholder,
        destination_shareholder.key(),
        TX_TYPE_SHARE_TRANSFER,
        shares,
        false,
        trade_date
    )?;
```

Figure 4.1: Excerpt of the definition of the `request_share_transfer` function

```
fn create_pending_transaction<'info>(
    transaction_pending: &mut Account<'info, TransactionPending>,
    money_market_fund: &mut Account<'info, MoneyMarketFund>,
    shareholder: &mut Account<'info, ShareHolder>,
    destination: Pubkey,
```

```
    tx_type: u64,
    amount: u64,
    self_service: bool,
    trade_date: i64,
) -> Result<()> {
    // ...
    transaction_pending.amount = amount;
```

Figure 4.2: Excerpt of the definition of the `create_pending_transaction` function

```
fn process_settlement<'info>(
    signer: &[&[&[u8]]],
    token_program: &mut Program<'info, Token2022>,
    mint: &mut Box<InterfaceAccount<'info, Mint>>,
    // authority: &mut Signer<'info>,
    token_account: &mut Box<InterfaceAccount<'info, TokenAccount>>,
    // fund_manager: &mut Account<'info, FundManager>,
    destination_token_account: &mut Option<InterfaceAccount<'info, TokenAccount>>,
    // shareholder: &mut Account<'info, ShareHolder>,
    transaction_pending: &mut Account<'info, TransactionPending>,
    date: i64,
    price: u64,
    // trade_date: i64,
) -> Result<()> {
    // ...
    let shares = match transaction_pending.tx_type {
        // ...
        TX_TYPE_SHARE_TRANSFER => {
            if let Some(destination_token_account) = destination_token_account {
                let shares = calculate_shares(transaction_pending.amount, price,
mint.decimals)?;
```

Figure 4.3: Excerpt of the definition of the `process_settlement` function

```
msg!("Share Amount: {}", shares);
```

Figure 4.4: Logging of the number of shares

Exploit Scenario

Alice requests a transfer of 100 shares to Bob. The number of shares logged is less than the expected 100. The off-chain code observes the incorrect amount, resulting in accounting errors.

Recommendations

Short term, correct the code in figures 4.1 through 4.4 so that the correct number of shares is logged. This will help prevent off-chain code from behaving incorrectly.

Long term, develop more thorough tests (e.g., for checking the correctness of log messages). More thorough tests likely would have revealed this bug.

5. remove_submitters can remove all submitters

Severity: High	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-FTSOLANA-5
Target: `solana-multisig/programs/solana-multisig/src/lib.rs`	

Description

The submitter count is validated incorrectly in the `remove_submitters` function, allowing to remove all submitters; effectively bricking the multisig.

The `remove_submitters` function checks that there is still one valid submitter left through:

```
require!(
    multisig.submitters.len() > 0,
    CustomError::InvalidSubmitterCount
);
```

Figure 5.1: `lib.rs#L165-L168`

However, this check is incorrect, and will always return `true`, as `submitters` is a fixed-size array:

```
pub struct MultisigConfig {
    pub submitters: [Pubkey; 10],
```

Figure 5.1: `lib.rs#L507-L508`

Exploit Scenario

The `remove_submitters` function is accidentally called with all submitters as parameters. As a result, the multsig has no valid submitter left and can no longer be used.

Recommendations

Short term, count how many submitters are nonzero instead of the submitter's array length:

```
multisig.submitters.iter().filter(|&&s| s !=
Pubkey::default()).count() > 0
```

Long term, improve the testing coverage, and ensure that every `require` condition is checked with an appropriate unit test.

6. is_frozen is checked on the wrong variable during the transfer of shares

Severity: **Medium**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-FTSOLANA-6
Target: `solana-multisig/programs/solana-multichain/src/lib.rs`	

Description
Incorrect variable usages allow frozen addresses to be usable when transferring shares.

The `RequestShareTransfer` and `RequestSelfServiceShareTransfer` functions use the shareholder source variable instead of the destination when checking if the address is frozen:

- `RequestShareTransfer` uses `shareholder` instead of `destination_shareholder`.

```
#[account(mut, has_one = money_market_fund @ ErrorCode::InvalidShareHolder,
constraint = shareholder.is_frozen == false @ ErrorCode::ShareholderFrozen)]
pub destination_shareholder: Account<'info, ShareHolder>,
```

Figure 6.1: solana-multichain/src/lib.rs#L1343-L1344

- `RequestSelfServiceShareTransfer` uses `shareholder` instead of `destination`.

```
#[account(mut, has_one = money_market_fund @ ErrorCode::InvalidShareHolder,
constraint = shareholder.is_frozen == false @ ErrorCode::ShareholderFrozen)]
pub destination: Account<'info, ShareHolder>,
```

Figure 6.2: solana-multichain/src/lib.rs#L1494-L1495

As a result, the transfer will succeed even if the destinations are frozen.

Exploit Scenario
The authorities contact Franklin Templeton to freeze Eve's account due to legal issues. Even though the account is frozen, Eve continues to receive shares from other accounts, leading to legal repercussions for Franklin Templeton.

Recommendations
Short term, perform the following actions:

- Check if the `destination_shareholder` is frozen in `RequestShareTransfer`.

- Check if the `destination` is frozen in `RequestSelfServiceShareTransfer`.

Long term, document the constraints that need to be applied to the different actors, highlighting the differences between actors with different levels of privileges. Add tests to ensure that every instruction properly prevents frozen accounts from interacting with the protocol.

7. Self-service functions can be called when the self-service is disabled

Severity: **Low**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-FTSOLANA-7
Target: `solana-multisig/programs/solana-multicain/src/lib.rs`	

Description

A missing validation on the `request_self_service_share_transfer` and `cancel_self_service_request` functions allows users to call these functions when the self-service is disabled.

Self-service functions can be enabled or disabled by the fund manager through `enable_self_service` and `disable_self_service`, respectively:

```
/// Enables the Self Service API
///
/// Note: Self Service allows shareholder accounts to call directly the associated
API
/// to create their own purchase, liquidation and cancellation requests
pub fn enable_self_service(ctx: Context<EnableSelfService>, _nonce: u64) ->
Result<()> {
    let fund_manager = &mut ctx.accounts.fund_manager;
    fund_manager.self_service_enabled = true;

    bump_nonce(fund_manager)?;
    Ok(())
}

/// Disables the Self Service API
///
/// Note: Self Service allows shareholder accounts to call directly the associated
API
/// to create their own purchase, liquidation and cancellation requests
pub fn disable_self_service(ctx: Context<DisableSelfService>, _nonce: u64) ->
Result<()> {
    let fund_manager = &mut ctx.accounts.fund_manager;
    fund_manager.self_service_enabled = false;

    bump_nonce(fund_manager)?;
    Ok(())
}
```

Figure 71: solana-multichain/src/lib.rs#L816-L838

All self-service functions use `only_when_self_service_enabled` to verify that they can be executed, except for `request_self_service_share_transfer` and `cancel_self_service_request`.

As a result, these functions can be called even after disabling the self-service feature.

Exploit Scenario

Eve finds a bug in `cancel_self_service_request`. To prevent exploitation of this bug, the fund manager disables the self-service functions through `disable_self_service`. However, Eve is still able to execute `cancel_self_service_request` and can continue to exploit the bug.

Recommendations

Short term, call `only_when_self_service_enabled` in `request_self_service_share_transfer` and `cancel_self_service_request`.

Long term, improve the test coverage and ensure that all of the self-service functions are tested in different conditions.

8. Frozen account can still cancel transactions

Severity: **Low**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-FTSOLANA-8
Target: `solana-multisig/programs/solana-multichain/src/lib.rs`	

Description
The `cancel_self_service_request` function does not check if the caller is frozen:

```
#[account(mut, has_one = money_market_fund @ ErrorCode::InvalidShareHolder,
constraint = shareholder.nonce == nonce @ ErrorCode::InvalidNonce)]
pub shareholder: Account<'info, ShareHolder>,
```

Figure 8.1: solana-multichain/src/lib.rs#L1601-L1602

As a result, frozen accounts can still cancel pending transactions.

This issue can be combined with TOB-FTSOLANA-1 to increase the impact of exploitation, as it will not be possible to freeze a shareholder that would cancel transactions from other users.

The same issue exists in `CancelRequest`.

Exploit Scenario
Eve's account is frozen. However, she can still interact with the program and cancel transactions.

Recommendations
Short term, check if the shareholder is frozen in `CancelSelfServiceRequest` and `CancelRequest`.

Long term, document the constraints that need to be applied to the different actors, highlighting the difference between actors with different levels of privileges. Add tests to ensure that all instructions properly prevent frozen accounts from interacting with the protocol.

9. Code duplication between recover_account and recover_asset

Severity: **Informational**	Difficulty: **Medium**
Type: Undefined Behavior	Finding ID: TOB-FTSOLANA-9
Target: `solana-multisig/programs/solana-multichain/src/lib.rs`	

Description

The `recover_account` and `recover_asset` functions differ in only a few lines. In particular:

- `recover_account` recovers all of a shareholder's tokens (highlighted in yellow in figure 7.1), as opposed to an amount specified in the instruction data.
- `recover_account` decrements the `MoneyMarketFund`'s `shareholder_accounts_counter` (highlighted in green in figure 7.2).

By having nearly identical implementations, one risks fixing a bug in one and but not the other.

```
pub fn recover_account(
    ctx: Context<RecoverAccount>,
    // trade_date: i64,
    _nonce: u64,
) -> Result<()> {
    let shareholder_shares = ctx.accounts.token_account.amount;
    [...]
    // Transfer all tokens from the shareholder's token account
    transfer_checked(
        CpiContext::new_with_signer(
            ctx.accounts.token_program.to_account_info(),
            TransferChecked {
                from: ctx.accounts.token_account.to_account_info(),
                mint: ctx.accounts.mint.to_account_info(),
                to: ctx.accounts.destination_token_account.to_account_info(),
                authority: ctx.accounts.mint.to_account_info(),
            },
            signer,
        ),
        shareholder_shares,
        ctx.accounts.mint.decimals,
    )?;
    [...]
    // Decrement the shareholder accounts counter
    ctx.accounts.money_market_fund.shareholder_accounts_counter = ctx
```

```
        .accounts
        .money_market_fund
        .shareholder_accounts_counter
        .saturating_sub(1);


    Ok(())
}
```

Figure 9.1: Excerpt of the definition of `recover_account`

Exploit Scenario

Alice, a Franklin Templeton developer, fixes a bug in `recover_account` but fails to realize the same bug exists in `recover_asset`. The bug persists in `recover_asset`.

Recommendations

Short term, implement `recover_account` by calling `recover_asset`. Specifically, pass the shareholder's full amount as the amount to be recovered. This will eliminate the duplicated code that currently exists between these two functions.

Long term, strive to reduce code duplication. Avoiding code duplication will help to avoid situations where a bug is fixed in one copy of a piece of code, but not in another.

10. Missing call to is_valid_submitter in AddSubmitters

Severity: **Informational**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-FTSOLANA-10
Target: `solana-multisig/programs/solana-multisig/src/lib.rs`	

Description

The AddSubmitters struct lacks a check that `payer` is an authorized submitter (figure 6.1). With the exception of `initialize`, all other `solana_multisig` instructions include this check (e.g., figure 6.2), suggesting the omission is an oversight.

```
#[derive(Accounts)]
pub struct AddSubmitters<'info> {
    #[account(mut, seeds = [b"multisig-config"], bump)]
    pub multisig: Account<'info, MultisigConfig>,
    #[account(mut, seeds = [multisig.key().as_ref()], bump)]
    pub multisig_rso: SystemAccount<'info>,
    pub payer: Signer<'info>,
    pub signer_0: Option<Signer<'info>>,
    pub signer_1: Option<Signer<'info>>,
    // ...
    pub signer_9: Option<Signer<'info>>,
}
```

Figure 10.1: The `AddSubmitters` struct, which does not check that `payer` is an authorized submitter

```
#[derive(Accounts)]
pub struct RemoveSubmitters<'info> {
    #[account(mut, seeds = [b"multisig-config"], bump)]
    pub multisig: Account<'info, MultisigConfig>,
    #[account(mut, seeds = [multisig.key().as_ref()], bump)]
    pub multisig_rso: SystemAccount<'info>,
    #[account(mut, constraint = is_valid_submitter(multisig.clone(), payer.key()) @
CustomError::InvalidSubmitter)]
    pub payer: Signer<'info>,
    pub signer_0: Option<Signer<'info>>,
    pub signer_1: Option<Signer<'info>>,
    // ...
    pub signer_9: Option<Signer<'info>>,
}
```

Figure 10.2: The `RemoveSubmitters` struct, which includes the necessary check

Exploit Scenario

Mallory prepares a transaction to add herself as a submitter. Mallory obtains the relevant signatures through phishing and social engineering. Mallory submits the transaction and becomes an authorized submitter.

Recommendations

Short term, add the missing check to `AddSubmitters`. This will ensure that only authorized submitters can submit transactions through `solana_multisig`.

Long term, expand `solana_multisig`'s tests to include negative tests. For example, a test of `add_submitters` where the payer is an unauthorized submitter could have uncovered this bug.

11. Bump seeds are not stored in PDAs

Severity: **Informational**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-FTSOLANA-11
Target: Various instructions	

Description

The Franklin Templeton programs use several types of PDAs (an example use appears in figure 5.1). Each use of one of these PDAs recomputes its bump seed. An unlucky deployment may occur in which computing one of the PDAs' bump seeds takes many iterations, making it cost-prohibitive to call an instruction with the PDA.

```
#[derive(Accounts)]
pub struct AddSubmitters<'info> {
    #[account(mut, seeds = [b"multisig-config"], bump)]
    pub multisig: Account<'info, MultisigConfig>,
    #[account(mut, seeds = [multisig.key().as_ref()], bump)]
    pub multisig_rso: SystemAccount<'info>,
    pub payer: Signer<'info>,
    pub signer_0: Option<Signer<'info>>,
    pub signer_1: Option<Signer<'info>>,
    // ...
    pub signer_9: Option<Signer<'info>>,
}
```

Figure 11.1: Example PDA used by the Franklin Templeton programs

Anchor's official documentation recommends storing a PDA's bump seed in the PDA. To give a specific example, a field `bump` could be added to the `MultisigConfig` struct, like in figure 5.2. Then, each use of `MultisigConfig` could be updated to use the `bump` field, like in figure 5.3.

```
#[account]
pub struct MultisigConfig {
    pub submitters: [Pubkey; 10],
    pub signers: [SignerKey; 10],
    pub high_threshold: u8,
    pub normal_threshold: u8,
    pub bump: u8,
}
```

Figure 11.2: Proposed modification of the `MultisigConfig` struct to store a bump seed

```
#[derive(Accounts)]
pub struct AddSubmitters<'info> {
    #[account(mut, seeds = [b"multisig-config"], bump = multisig.bump)]
    pub multisig: Account<'info, MultisigConfig>,
    #[account(mut, seeds = [multisig.key().as_ref()], bump)]
    pub multisig_rso: SystemAccount<'info>,
    pub payer: Signer<'info>,
    pub signer_0: Option<Signer<'info>>,
    pub signer_1: Option<Signer<'info>>,
    // ...
    pub signer_9: Option<Signer<'info>>,
}
```

Figure 11.3: Proposed modification of the `AddSubmitters` struct to use the `MultisigConfig`'s bump seed

Exploit Scenario

An unlucky deployment causes the `solana_multisig` program to receive a program ID that, when hashed with "multisig-config" and a bump seed, takes many iterations to find a bump seed that is not on the ED25519 curve. Each time the PDA is passed to an instruction, the steps to find the bump seed are repeated. The cost to use the Franklin Templeton programs is higher than necessary.

Recommendations

Short term, store PDAs' bump seeds in the PDA. This will reduce the cost of using the PDAs. Moreover, it will help protect against unlucky deployments where computing one of the PDAs' bump seeds requires many iterations.

Long term, when creating PDAs, keep the following in mind:

- Create PDAs only with canonical bump seeds. Creating PDAs with noncanonical bump seeds could allow an attacker to create multiple PDAs for the same set of non-bump seeds.
- Avoid the following situation:
    - Part of a PDA's seeds are user-controlled.
    - The PDA's bump seed is not stored (i.e., it is recomputed each time the PDA is used).
    - The user that causes a PDA to be created is not the same user that must pass the PDA in an instruction.

In such a situation, Mallory could try to choose seeds for which it is expensive to compute the PDA's bump seed. If Mallory can then require Alice to pass the PDA in an instruction, Mallory can cause Alice to waste lamports computing the PDA's bump seed.

References

- Anchor PDA Constraints
- Solana Beta: Necessary to pass bump for PDA in the instruction data when using Anchor?
- Solana Beta: Difference between bump vs account.bump in anchor

12. API key exposure in configuration files

Severity: Medium	Difficulty: **Low**
Type: Data Exposure	Finding ID: TOB-FTSOLANA-12
Target: `solana-multisig/Anchor.toml` and `solana-multichain/Anchor.toml`	

Description
The Solana RPC API key is hardcoded in the `Anchor.toml` file:

```
[provider]
cluster = "https://devnet.helius-rpc.com/?api-key=********************************"
```

Figure 12.1: `solana-multisig/Anchor.toml#L16-L17`

This exposes the key, allowing unauthorized usage of the Helius RPC service.

Exploit Scenario
An attacker can retrieve the exposed key, abuse it to overload the RPC service, or collect sensitive network data.

Recommendations
Short term, use the `--provider.cluster` flag to provide the API key, and set the API key in an environment variable.

Long term, enable GitHub secret scanning, or integrate a similar service if you do not use GitHub.

13. solana_multisig uses an outdated version of anchor-lang

Severity: **Informational**	Difficulty: **High**
Type: Patching	Finding ID: TOB-FTSOLANA-13
Target: `solana_multisig/Cargo.toml`	

Description

The `solana_multisig` program uses anchor-lang version 0.29.0; however, the most recent version is 0.30.1. Since silent bug fixes are common, projects should strive to use the most recent versions of their dependencies whenever possible.

```
[dependencies]
anchor-lang = "0.29.0"
```

Figure 13.1: `anchor-lang` version used by `solana_multisig` (as of January 21, 2025)

Exploit Scenario

Mallory finds a bug in anchor-lang version 0.29.0. Mallory exploits the flaw in `solana_multisig` knowing that it relies on the outdated, vulnerable dependency.

Recommendations

Short term, upgrade `solana_multisig`'s `anchor-lang` requirement to version 0.30.1. This will allow `solana_multisig` to use the most up-to-date version of `anchor-lang`.

Long term, enable a service such as Dependabot to regularly notify you of updates. This will help keep your dependencies up to date.

14. Multiple tautologies make the checks always return true

Severity: Informational	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-FTSOLANA-14
Target: `solana-multisig/programs/solana-multisig/src/lib.rs`	

Description

Several checks on `MAX_THRESHOLD` are incorrect and always return true.

`MAX_THRESHOLD` is a `u8` equal to the type maximum's value:

```
const MAX_THRESHOLD: u8 = u8::MAX;
```

Figure 14.1: solana-multisig/src/lib.rs#L1

`MAX_THRESHOLD` is then checked to be greater or equal to other `u8` values:

```
require!(
    high > 0 && high <= MAX_THRESHOLD,
    CustomError::InvalidThreshold
);
```

Figure 14.2: solana-multisig/src/lib.rs#L247-L250

```
require!(
    normal > 0 && normal <= MAX_THRESHOLD,
    CustomError::InvalidThreshold
);
```

Figure 14.3: solana-multisig/src/lib.rs#L285-L288

However, given it has the maximum value of the type, `<= MAX_THRESHOLD` always returns `true`.

Recommendations

Short term, either remove the check on the max threshold, or set it to a correct value.

Long term, improve the testing coverage, and ensure that every `require` condition is checked with an appropriate unit test. Add linters to your codebase; this issue is detected by Clippy.

15. No evidence of linter usage

Severity: **Informational**	Difficulty: **High**
Type: Patching	Finding ID: TOB-FTSOLANA-15
Target: entire codebase	

Description
Running Clippy over the codebase produces many warnings (table 15.1), suggesting it is not used. Clippy is an official Rust tool and should be used by every Rust project.

Lint name	Number of warnings/errors
`absurd_extreme_comparisons`	2
`bool_comparison`	17
`derivable_impls`	1
`extra_unused_lifetimes`	6
`len_zero`	1
`needless_borrow`	4
`needless_late_init`	2
`needless_lifetimes`	4
`too_many_arguments`	2
`unnecessary_fallible_conversions`	8

Table 15.1: Clippy lints that issue warnings on the Franklin Templeton codebase, along with the number of warnings for each lint

Exploit Scenario
Mallory runs Clippy over the Franklin Templeton programs and notices that one of the warnings is not benign. Mallory develops an exploit for the underlying bug and steals funds from the Franklin Templeton contracts.

Recommendations

Short term, run Clippy over the codebase and address each of the warnings that it issues. This will reduce the likelihood that the code contains bugs.

Long term, incorporate Clippy into your continuous development process. For example, before each change is made to the code's main branch, ensure that Clippy produces no warnings. This will help to ensure that the code remains free of bugs.

16. Mix of debugging and production code

Severity: Informational	Difficulty: **High**
Type: Patching	Finding ID: TOB-FTSOLANA-16
Target: `solana-multisig/programs/*`	

Description

The codebase contains several commented operations and checks, and it is unclear whether they are meant to be included in the production code.

These operations include the following:

- Debugging information:

```
// msg!("calculate shares");
let upscaled_amount: u128 = amount.try_into().unwrap();
// msg!("upscaled amount: {}", upscaled_amount);
let upscaled_price: u128 = price.try_into().unwrap();
// msg!("upscaled price: {}", upscaled_price);
```

Figure 16.1: solana-multichain/src/lib.rs#L1723-L1727

- Temporary access controls:

```
#[account(mut,/*address =
pubkey!("2B6QUdfYHZq9vwTmHfcmc3CmJ1W7A2XLVeWjbkgE6Htm")*/)]
pub authority: Signer<'info>,
```

Figure 16.2: solana-multichain/src/lib.rs#L1014-L1015

- Remaining TODOs:

```
// TODO: Should probably just make this account optional instead of using a dummy
account
let destination_token_account = &mut ctx.accounts.destination_token_account;
```

Figure 16.3: solana-multichain/src/lib.rs#L450-L451

The presence of several comments and TODOs indicates that the codebase may require changes before deployment. It is unclear if some of the operations will be left uncommented.

Recommendations

Short term, do not mix work-in-progress code with production code. Use `#[cfg(test)]` for testing statements that do not need to be in the deployed code.

Long term, create and follow stricter software development practices. This can include:

- Enforcing peer review on every PRs
- Integrating linters in the CI
- Adopting and enforcing a consistent coding style
- Establishing a well-defined branching strategy
- Enforcing minimal unit test coverage
- Scheduling regular refactoring and documentation update sessions

17. Lack of documentation	
Severity: **Informational**	Difficulty: **High**
Type: Patching	Finding ID: TOB-FTSOLANA-17
Target: Entire codebase	

Description
The project lacks documentation, which inhibits the code's readability and increases the risk of users misusing the programs.

The project should have a `README` with at least the following information:

- Instructions for building the project
- Instructions for running the built artifacts
- Instructions for running the project's tests

Additionally, the project should have documentation with at least the following information:

- A description of the project's two programs, `solana_multisig` and `solana_multichain`
- Descriptions of the actors expected to interact with the programs
- Descriptions of the actions each actor can perform
- Descriptions of the accounts the programs maintain and how they relate to one another, with a focus on their related access controls
- A diagram showing the lifecycle of transactions
- Descriptions of the features present for future needs
- The share of dividends and the expectations on the accounts' frozen state
- The expected difference between AIP and cash purchase

Exploit Scenario
Alice attempts to build and deploy her own copy of the Franklin Templeton Benji contracts. Without instructions, Alice deploys them incorrectly. Users of Alice's copy of the contracts suffer financial loss.

Recommendations

Short term, add documentation to the project with the minimal information listed above. This will help users to build, run, and test the project, and will reduce the likelihood that they will misuse it.

Long term, as the project evolves, ensure that the documentation is updated so it provides value to users.

18. Insufficient test coverage	
Severity: **Informational**	Difficulty: **High**
Type: Testing	Finding ID: TOB-FTSOLANA-18
Target: Entire codebase	

Description
The `solana_multichain` program features 23 instructions. However, only 15 of them are tested. To the extent possible, every line of every function that executes on-chain should be tested.

The following `solana_multichain` instructions are currently untested:

- `cancel_request`
- `cancel_self_service_request`
- `disable_self_service`
- `freeze_shareholder`
- `remove_shareholder`
- `request_self_service_share_transfer`
- `request_share_transfer`
- `Unfreeze_shareholder`

Exploit Scenario
Mallory finds a bug in an untested `solana_multichain` instruction and develops an exploit for it. The bug would likely have been found by Franklin Templeton through more thorough testing.

Recommendations
Short term, write tests for each of the instructions in the bulleted list above. Specifically, write tests to ensure that the instruction works correctly under ideal conditions, and that it returns appropriate errors under all other conditions. In particular, we recommend improving the tests (both for positive and negative tests) for:

- The pending transactions' lifecycle
- The self service functions
- The frozen status of shareholders
- The multisig addition and removal of signers and submitters
- Every `require` condition within the codebase

Long term, regularly review your project's test coverage to ensure that all important conditions are tested. This will help ensure that the tests stay up to date with the code.

19. Insufficient logging

Severity: **Informational**	Difficulty: **High**
Type: Error Reporting	Finding ID: TOB-FTSOLANA-19
Target: Entire codebase	

Description

The `solana_multisig` and `solana_multichain` programs feature seven and 23 instructions, respectively. Only three of the latter perform direct logging. Programs should emit log messages from successful instructions to aid off-chain code.

The following `solana_multisig` instructions do not directly emit log messages:

- `initialize`
- `add_submitters`
- `remove_submitters`
- `update_signers`
- `update_high_threshold`
- `update_normal_threshold`
- `execute_instruction`

The following `solana_multichain` instructions do not directly emit log messages:[1]

- `initialize`
- `distribute_dividend`
- `settle_transaction`
- `set_up_aip`
- `request_cash_purchase`
- `request_cash_liquidation`
- `request_full_liquidation`
- `request_share_transfer`
- `cancel_request`
- `enable_self_service`
- `disable_self_service`
- `request_self_service_cash_purchase`
- `request_self_service_cash_liquidation`
- `request_self_service_full_liquidation`

[1] The three instructions not in this list are `recover_account`, `recover_asset`, and `adjust_shares`.

- `request_self_service_share_transfer`
- `cancel_self_service_request`
- `add_shareholder`
- `remove_shareholder`
- `freeze_shareholder`
- `unfreeze_shareholder`

Exploit Scenario

Alice, a Franklin Templeton client, submits an instruction to the Franklin Templeton programs. The instruction is completed successfully. However, Alice has difficulty determining the path that the instruction took. Alice's time would have been saved had the instruction emitted proper log messages.

Recommendations

Short term, identify and document which operations require events for monitoring and create an incident response plan accordingly. Having a well-defined events strategy will ease the integration of monitoring solutions and will help verifying the program's correct behavior.

Long term, if new instructions are added to the programs, ensure that they also adhere to this standard of directly emitting log messages. Test your incident response plan with a drill exercise.

20. Incorrect fix pushed mid-review

Severity: **Informational**	Difficulty: **High**
Type: Patching	Finding ID: TOB-FTSOLANA-20
Target: Entire codebase	

Description

After we reported TOB-FTSOLANA-3 during the review, Franklin Templeton provided a fix for the issue, but it is incorrect.

TOB-FTSOLANA-3 is related to an incorrect access control on the `freeze_shareholder` function. The fix provided by Franklin Templeton did not address the issue, nor does the fix update the affected function:

```
991  +         /// Force remove a shareholder in case of user trying to spam the
       system.
992  +         pub fn force_remove_shareholder(ctx:
       Context<ForceRemoveShareholder>) -> Result<()> {
993  +             let money_market_fund = &mut ctx.accounts.money_market_fund;
994  +             bump_nonce(money_market_fund)?;
995  +             let shareholder = &mut ctx.accounts.shareholder;
996  +             let token_account = &mut ctx.accounts.token_account;
997  +             let shareholder_shares = token_account.amount;
998  +
999  +             only_when_shareholder_has_no_shares(shareholder_shares)?;
1000 +             only_when_shareholder_has_no_pending_transactions(shareholder)?;
1001 +
1002 +             money_market_fund.shareholder_accounts_counter =
       money_market_fund
1003 +                 .shareholder_accounts_counter
1004 +                 .saturating_sub(1);
1005 +
1006 +             Ok(())
1007 +         }
1008 +
```

Figure 20.1: Fix provided for TOB-FTSOLANA-3

Although reviewing fixes is part of code review, this fix targeting the wrong function highlights the need for stricter code practices to prevent incorrect updates.

Recommendations

Short term, apply the recommendations provided in TOB-FTSOLANA-3.

Long term, create and follow stricter software development practices. This can include the following:

- Enforcing peer review on every PR
- Integrating linters in the CI
- Adopting and enforcing a consistent coding style
- Establishing a well-defined branching strategy
- Enforcing minimal unit test coverage
- Scheduling regular refactoring and documentation update sessions

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage
Transaction Ordering	The system's resistance to transaction-ordering attacks

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.

Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Non-Security-Related Recommendations

The following recommendations are not associated with specific vulnerabilities. However, implementing them may enhance code readability and prevent the introduction of vulnerabilities in the future.

Multisig

1. **Rewrite the code in Figure C.1 to hold `last_signer_index` across loop iterations, rather than `last_recovered_signer`.** Currently, each loop iteration performs two linear searches: one to determine `current_signer_index`, and one to recover `last_signer_index`. By working directly with indices rather than the keys, these additional searches can be avoided.

```
match multisig
    .signers
    .iter()
    .position(|&x| x.key == recovered_signer.key)
{
    Some(index) => current_signer_index = index,
    None => return err!(CustomError::InvalidSignature),
}
match multisig
    .signers
    .iter()
    .position(|&x| x.key == last_recovered_signer.key)
{
    Some(index) => last_signer_index = index,
    None => return err!(CustomError::InvalidSignature),
}
```

Figure C.1: Linear searches performed during each loop iteration in `is_threshold_enough`

2. **Make `payer` mutable in `AddSubmitters`.** Adding `#[account(mut)]` will explicitly state that this account is assumed to pay the fees.

3. **Remove the unused import in Figure C.2 from both `solana-multichain/src/lib.rs` and `solana-multichain/src/lib.rs`.**

```
use solana_program::pubkey;
```

Figure C.2: Unused import in both `solana-multichain/src/lib.rs` and `solana-multichain/src/lib.rs`

4. **Replace the code in Figure C.3 with that of figure C.4.** The latter is simpler and more concise.

```
for s in multisig.submitters.iter() {
    if *s == submitter {
        return true;
    }
}
false
```

Figure C.3: Code in `solana-multichain/src/lib.rs` that could be simplified

```
multisig.submitters.iter().any(|&s| s == submitter)
```

Figure C.4: Proposed rewrite of the code in figure C.3

5. **Remove the "8 +" from the code in figure C.5, as the 8 bytes for the Anchor discriminators are already included in the structs' sizes (figures C.6–7).**

```
#[derive(Accounts)]
#[instruction(_token_name: String, _token_symbol: String, _token_uri: String,
token_decimals: u8)]
pub struct Initialize<'info> {
    #[account(init, payer = authority, space = 8 + FundManager::LEN)]
    pub fund_manager: Account<'info, FundManager>,
    #[account(init, payer = authority, space = 8 + MoneyMarketFund::LEN)]
```

Figure C.5: Excerpt of the definition of `Initialize`

```
impl FundManager {
    const LEN: usize = 8 + 32 + 32 + 32 + 1 + 8;
}
```

Figure C.6: Definition of `FundManager::SIZE`

```
impl MoneyMarketFund {
    const LEN: usize = 8 + 32 + 8 + 8 + 8 + 8 + 8 + 8;
}
```

Figure C.7: Definition of `MoneyMarketFund::SIZE`

6. **Eliminate the use of the ineffective test construct in figure C.8.** The `assert.fail` will get caught by the `catch (err)` rather than cause the test to fail. Instead, set a flag after the call to `fetch` (to indicate success) or in the `catch` block (to indicate failure), and check the flag after the `try-catch` statement.

```
 try {
    await program.account.shareHolder.fetch(shareholder);
    assert.fail("Shareholder account should be closed");
  } catch (err) {
    // console.log("Shareholder account successfully closed.");
  }
});
```

Figure C.8: Ineffective test construct in `solana-multichain/tests/multichain-solana.ts`

7. **In the call to `create_pending_transaction` in `request_self_service_share_transfer`, change the second to last argument to `true`.** The argument indicates whether the request is a self-service request. The argument is currently incorrectly set to `false`.

```
create_pending_transaction(
    transaction_pending,
    money_market_fund,
    shareholder,
    destination.key(),
    TX_TYPE_SHARE_TRANSFER,
    shares,
    false,
    trade_date
)?;
```

Figure C.9: The call to `create_pending_transaction` in `request_self_service_share_transfer`

8. **In `solana-multichain/src/lib.rs`, wherever there is a `destination_token_account`, add a constraint that the account's `mint` is the expected one.** Even if this check is performed by the token program, adding the constraint will make it explicit and catch invalid accounts sooner.

```
#[derive(Accounts)]
#[instruction(tx_id: Pubkey, _date: i64, _price: u64, nonce: u64)]
pub struct SettleTransaction<'info> {
    // ...
    #[account(
        mut,
        token::token_program = token_program,
        token::mint = mint,
    )]
```

```
    pub destination_token_account: Option<InterfaceAccount<'info, TokenAccount>>,
    // ...
}
```

Figure C.10: Proposed additional check for destination token accounts

General

9. **Eliminate the uses of checked arithmetic that are immediately followed by a call to `unwrap`.** An example appears in figure C.11. Immediately unwrapping a checked arithmetic operation provides no benefits over enabling overflow checks.

```
acquired_threshold = acquired_threshold
    .checked_add(recovered_signer.weight)
    .unwrap();
```

Figure C.11: A call to `checked_add` whose result is immediately unwrapped

10. **Consistently use SIZE or LEN for the size of structs.** Currently, both are used.

```
impl MultisigConfig {
    // added a buffer at the end to account for the padding
    const SIZE: usize = 8 + (32 * 10) + (33 * 10) + 1 + 1;
}
```

Figure C.12: A constant `SIZE` used for the size of a struct

```
impl FundManager {
    const LEN: usize = 8 + 32 + 32 + 32 + 1 + 8;
}
```

Figure C.13: A constant `LEN` used for the size of a struct

11. **Deduplicate test names.** For each of the following names, there are two tests in `solana-multichain/tests/multichain-solana.ts` with that name.

    - `should settle a pending liquidation request for the given account`
    - `should settle a pending full liquidation request for the given account`

12. **Enable `noUnusedLocals` in the project's `tsconfig.json` file.** The project's test file has many unused locals. An example appears in figure C.14.

```
// Only get the destination wallet it its ANOTHER actual shareholder wallet
let destinationTokenAccount = DEFAULT_KEY;
if (transaction.account.destination.toString() != DEFAULT_KEY.toString()) {
  const shareholderAccountInfo = await
program.account.shareHolder.fetch(shareholder);
  const destinationShareholderWallet = (await
program.account.shareHolder.fetch(transaction.account.destination)).wallet;
  destinationTokenAccount = getAssociatedTokenAddressSync(
    mint, destinationShareholderWallet, true, TOKEN_2022_PROGRAM_ID
  );
```

```
}
```

Figure C.14: An example of an unused local in `solana-multichain/tests/multichain-solana.ts`

13. **Capture the error on unwrap() operations.** Several operations call `unwrap()` without the question mark operator. This pattern makes the code more difficult to debug and is overall error-prone.

14. **Review the structure fields to ensure they do not include unused fields.** For example, the clock Sysvar is present in several structures, but it is not needed.

D. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

From February 3 to February 5, 2025, Trail of Bits reviewed the fixes and mitigations implemented by the Franklin Templeton team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the 20 issues described in this report, Franklin Templeton has resolved 19 issues, and has not resolved the remaining one issue. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	cancel_self_service_request can be called on any pending transaction	Resolved
2	Closing and reopening pending transactions allows to execute malicious actions	Resolved
3	Shareholder can prevent being frozen by increasing their nonce	Resolved
4	Incorrect logging of transferred shares	Resolved
5	remove_submitters can remove all the submitters	Resolved
6	is_frozen is checked on the wrong variable during the transfer of shares	Resolved
7	Self-service functions can be called when the self-service is disabled	Resolved
8	Frozen account can still cancel transactions	Resolved
9	Code duplication between recover_account and recover_asset	Unresolved
10	Missing call to is_valid_submitter in AddSubmitters	Resolved
11	Bump seeds are not stored in PDAs	Resolved

12	API Key exposure in configuration files	Resolved
13	solana_multisig uses an outdated version of anchor-lang	Resolved
14	Multiple tautologies make the checks always return true	Resolved
15	No evidence of linter usage	Resolved
16	Mix of debugging and production code	Resolved
17	No documentation	Resolved
18	Insufficient test coverage	Resolved
19	Insufficient logging	Resolved
20	Incorrect fix pushed mid-review	Resolved

Detailed Fix Review Results

TOB-FTSOLANA-1: cancel_self_service_request can be called on any pending transaction
Resolved. `transaction_pending.shareholder == shareholder.key()` was added to `CancelSelfServiceRequest` and `CancelRequest`.

TOB-FTSOLANA-2: Closing and reopening pending transactions allows to execute malicious actions
Resolved. Franklin Templeton aims to mitigate the issue with the usage of nonce on the shareholder and the offchain code's logic.

We still recommended that Franklin Templeton implement stronger validation on all the transaction's fields (e.g., by keeping and comparing a hash of the structure's fields).

TOB-FTSOLANA-3: Shareholder can prevent being frozen by increasing their nonce
Resolved. The `force_freeze_shareholder` instruction was added. This function behaves similarly to `freeze_shareholder` without the nonce verification.

TOB-FTSOLANA-4: Incorrect logging of transferred shares
Resolved. The code logic was updated.

We recommended that Franklin Templeton add an inline comment on the usage of `share` versus `amount` in `request_self_service_share_transfer` directly, given that the current structure may lead to confusion when updating the code.

TOB-FTSOLANA-5: remove_submitters can remove all the submitters
Resolved. The check in `remove_submitters` was updated to count the number of empty public keys.

TOB-FTSOLANA-6: is_frozen is checked on the wrong variable during the transfer of shares
Resolved. The check is now applied to the correct variable.

TOB-FTSOLANA-7: Self-service functions can be called when the self-service is disabled
Resolved. The `only_when_self_service_enabled` function is now called in `request_self_service_share_transfer` and `cancel_self_service_request`.

TOB-FTSOLANA-8: Frozen account can still cancel transactions
Resolved. A check was added to ensure that the shareholder is frozen in `CancelSelfServiceRequest`. The check was not applied to `CancelRequest`, as it was the expected behavior.

TOB-FTSOLANA-9: Code duplication between recover_account and recover_asset
Unresolved. The duplicated code was still present during the fix review.

TOB-FTSOLANA-10: Missing call to is_valid_submitter in AddSubmitters
Resolved. `is_valid_submitter` was added to `AddSubmitters.`

TOB-FTSOLANA-11: Bump seeds are not stored in PDAs
Resolved. The bump seeds are now stored in the PDAs.

TOB-FTSOLANA-12: API Key exposure in configuration files
Resolved. The API key was removed, and Franklin Templeton stated that the API Key was for debugging purposes only and that its leakage would not have impacted the project.

TOB-FTSOLANA-13: solana_multisig uses an outdated version of anchor-lang
Resolved. The dependency was updated.

TOB-FTSOLANA-14: Multiple tautologies make the checks always return true
Resolved. `MAX_THRESHOLD` was set to 250.

TOB-FTSOLANA-15: No evidence of linter usage
Resolved. The majority of Clippy warnings were addressed. The remaining warnings were still present:

```
warning: manual implementation of an assign operation
   --> programs/solana-multisig/src/lib.rs:540:9

warning: this function has too many arguments (8/7)
    --> programs/solana-multichain/src/lib.rs:1835:1

warning: this function has too many arguments (8/7)
    --> programs/solana-multichain/src/lib.rs:1903:1
```

Figure D.1: Remaining Clippy warnings

TOB-FTSOLANA-16: Mix of debugging and production code
Resolved. We noticed that one additional TODO was added in the codebase; however, Franklin Templeton stated that this TODO will be removed before deployment.

TOB-FTSOLANA-17: No documentation
Resolved. A readme with a description of the programs was added. We encourage Franklin Templeton to continue the effort of documentation, in particular around the Solana-specific features.

TOB-FTSOLANA-18: Insufficient test coverage
Resolved. 35 tests were added to the multichain program. However, none were added to the multisig program.

To ease the testing on different setups, we recommend the following change to the `Anchor.toml` file of the multisig program:

```
--- a/solana-multisig/Anchor.toml
+++ b/solana-multisig/Anchor.toml
@@ -15,7 +15,7 @@ url = "https://api.apr.dev"

 [provider]
 cluster = "localnet"
-wallet = "/home/node/.config/solana/id.json"
+wallet = "~/.config/solana/id.json"
```

Figure D.2: Diff on `solana-multisig/Anchor.toml`

We recommend that Franklin Templeton continue to add tests within the codebase.

TOB-FTSOLANA-19: Insufficient logging
Resolved. Franklin Templeton stated the following:

> *We do not need to add extra log messages to our program as we have an internal indexer for parsing all events from Franklin Templeton programs. This is an explicit design choice allowing us to keep our CU's down in our main programs. It also has the*

added benefit of making sure our source of truth is always the solana chain state and not user influenced. We opted to only log things we could not compute from that state.

Internally our parser lookes at every transaction that has to do with the Franklin Templeton programs and parses out all individual ix's using the provided anchor coder / decoders. We keep a log of all events and accounts related to those events as well as their balances and onchain state from before and after each tx and ix in our own db. Its actually a requirement for verifying the history of actions preformed on the fund and we can verify any account activity by rerunning it through the parser logic for any specific shareholder as well.

TOB-FTSOLANA-20: Incorrect fix pushed mid-review
Resolved. The incorrect fix was removed.

E. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.