Security Audit Report

MetaMask Card Baanx Withdraw Program

v1.0

June 20, 2025

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by Baanx Group Ltd. to perform a security audit of Baanx Withdraw Solana Program for MetaMask Card.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/Baanx-Group/anchor-withdraw
Commit	`083780392aa0fffff4764ab62d0fa403d6990971`
Scope	The scope of the audit is restricted to the `contracts/anchor/withdraw/programs/withdraw` directory, excluding the cryptographic library in `contracts/anchor/withdraw/programs/withdraw/src/utils/ed25519.rs`.
Fixes verified at commit	`80e3475ca7643d753ad81d62930d06cdfe7ed4a9`

	Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The MetaMask Card Baanx Withdraw program is a fund management system on Solana that provides two primary functions: a withdrawal mechanism that allows an authorized operator to transfer tokens from accounts to a beneficiary, and a multi-send feature for distributing tokens from a treasury to multiple recipients.

The program implements role-based access (admin, operator, beneficiary), reentrancy protection, signature verification for multi-send operations, and support for both SPL Token and Token-2022 standards.

It includes administrative functions for key management and a pause mechanism for emergency response.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, **Partially Resolved**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase’s code quality criteria as follows:

Criteria	Status	Comment
Code complexity	**Medium**	-
Code readability and clarity	**Medium-High**	-
Level of documentation	**Medium-High**	-
Test coverage	**Medium-High**	Test coverage is analyzed in detail in the Appendix.

Summary of Findings

No	Description	Severity	Status
1	Ineffective pause mechanism disables emergency control in the `multi_send` instruction	Major	Resolved
2	Missing initialization of reentrancy guard in the withdraw instruction	Minor	Resolved
3	Incorrect reset of reentrancy guard on transfer validation failure	Minor	Resolved
4	Ineffective timeout mechanism for reentrancy guard	Minor	Resolved
5	Incorrect comparison operator in reentrancy guard timeout mechanism	Minor	Resolved
6	Missing decimals validation in `validate_token_transfer` function	Minor	Resolved
7	Missing public key validation	Minor	Resolved
8	Input array misalignment risk enables erroneous token transfers	Minor	Resolved
9	Missing frozen-account check in token validation	Minor	Resolved
10	Integer overflow in batch status tracking masks transfer failures	Minor	Resolved
11	Fixed signature index prevents priority fee instructions	Informational	Resolved
12	Redundant reentrancy guard reset before error return	Informational	Resolved
13	The initialization of the program can be front-run	Informational	Resolved
14	Inefficient logging with `msg!` macro	Informational	Resolved
15	Inconsistent amount check bypasses `validate_transfer` flag	Informational	Resolved
16	Programs should implement a two-step ownership transfer	Informational	Acknowledged
17	Miscellaneous comments	Informational	Resolved

Detailed Findings

1. Ineffective pause mechanism disables emergency control in the `multi_send` instruction

Severity: Major

In `contracts/anchor/withdraw/programs/withdraw/src/instructions/multi_send.rs`, the `multi_send` instruction performs a conditional check intended to enforce an emergency pause mechanism via `ctx.accounts.program_config.paused`.

However, this `program_config` points to the `MultiSendConfig` struct, whose `paused` field is set to `false` during initialization and lacks any mechanism for subsequent modification.

Unlike the withdrawal logic, which correctly employs a mutable `ProgramConfig.paused` field that can be toggled using the `set_pause` instruction, the `MultiSendConfig` design does not provide an interface to alter the pause state post-deployment. As a result, the pause check in `multi_send` is effectively non-functional; the instruction always executes regardless of security context or administrative intent.

This renders the `multi_send` path immune to emergency shutdown procedures, eliminating a key operational safeguard. In the event of a detected threat or vulnerability, there is no mechanism to halt token transfers initiated through `multi_send`, significantly undermining the protocol's incident response capabilities.

Recommendation

We recommend aligning `MultiSendConfig` with the withdrawal module's design by introducing a mutable paused field and an associated administrative instruction to toggle it.

Status: Resolved

2. Missing initialization of reentrancy guard in the `withdraw` instruction

Severity: Minor

The `ctx.accounts.reentrancy_guard.initialized` field is not set to `true` after activating the reentrancy guard in the `withdraw` function in `contracts/anchor/withdraw/programs/withdraw/src/instructions/withdraw.rs:42-46`.

This omission could lead to inconsistent state management and potential vulnerabilities due to reliance on the `initialized` field elsewhere in the program.

Recommendation

We recommend setting `ctx.accounts.reentrancy_guard.initialized` to `true` in line `47` to properly initialize the reentrancy guard.

Status: Resolved

3. Incorrect reset of reentrancy guard on transfer validation failure

Severity: Minor

The reentrancy guard, enforced by the `reentrancy_guard.is_processing` and `reentrancy_guard.initialized` fields, is reset when a transfer validation fails in `contracts/anchor/withdraw/programs/withdraw/src/instructions/withdraw.rs:128-130` and `contracts/anchor/withdraw/programs/withdraw/src/instructions/multi_send.rs:218-220`.

However, this validation failure is not resulting in an error, and the loop continues to process subsequent transfers.

Resetting the reentrancy guard prematurely leaves the remaining transfers unprotected, potentially exposing the function to reentrancy attacks.

Recommendation

We recommend removing the reentrancy guard reset logic from the transfer validation failure block to ensure the guard remains active throughout the entire loop.

Status: Resolved

4. Ineffective timeout mechanism for reentrancy guard

Severity: Minor

The timeout mechanism for `reentrancy_guard.initialized` in `contracts/anchor/withdraw/programs/withdraw/src/state.rs:197-202` will not be effective if `reentrancy_guard.initialized` is stuck at `true`.

In such cases, `reentrancy_guard.is_processing` will also likely remain stuck at `true`, preventing further operations indefinitely and rendering the timeout mechanism ineffective.

Recommendation

We recommend introducing a recovery mechanism to reset `reentrancy_guard.is_processing` and `reentrancy_guard.initialized` after the timeout period has elapsed, ensuring the system can recover from such a state.

Status: Resolved

5. Incorrect comparison operator in reentrancy guard timeout mechanism

Severity: Minor

The timeout mechanism for the reentrancy guard uses a comparison to check if the elapsed time since `reentrancy_guard.timestamp` is less than `300` seconds in `contracts/anchor/withdraw/programs/withdraw/src/state.rs:197-202`.

This prevents the timeout mechanism from becoming effective after `300` seconds, as the condition will never allow recovery from a stuck reentrancy guard.

Recommendation

We recommend replacing the < operator with a > operator to ensure the timeout mechanism becomes effective after `300` seconds, e.g.

`Clock::get()?.unix_timestamp - reentrancy_guard.timestamp > 300`.

Status: Resolved

6. Missing decimals validation in `validate_token_transfer` function

Severity: Minor

The `validate_token_transfer` function, defined in `contracts/anchor/withdraw/programs/withdraw/src/utils/validation.rs:10-118`, does not include a `decimals` parameter to verify the provided decimals against the decimals of the token mint.

A mismatch in decimals could lead to an unexpected error during a subsequent `transfer_checked` operation in the `withdraw` and `multi_send` functions.

Recommendation

We recommend adding a `decimals` parameter to the `validate_token_transfer` function and including a validation step to ensure the provided decimals match the decimals of the token mint.

Status: Resolved

7. Missing public key validation

Severity: Minor

The `initialize` instruction in `contracts/anchor/withdraw/programs/withdraw/src/instructions/initialize.rs` accepts several public keys, including `operator`, `beneficiary`, `admin`, `multi_send_operator`, `signer`, and `treasury`, without performing any validation checks.

Similarly, administrative instructions such as `set_beneficiary`, `set_operator`, `set_signer`, `set_admin`, `set_treasury`, and `set_multi_send_operator` do not validate the provided public keys.

This lack of input validation enables the assignment of roles to incorrect or zero addresses.

Recommendation

We recommend implementing strict validation for each provided public key during initialization and administrative updates.

At a minimum, ensure that the public keys are not equal to `Pubkey::default()`.

Status: Resolved

8. Input array misalignment risk enables erroneous token transfers

Severity: Minor

The `withdraw` and `multi_send` instructions utilize parallel arrays for `amounts`, `decimals`, and `remaining_accounts` without explicit structuring or validation of their relationships.

The program relies on implicit positional alignment, indexing `remaining_accounts` at `i*3`, `i*3+1`, and `i*3+2` to access the `source`, `mint`, and `destination` accounts, respectively.

This design offers no intrinsic linkage between the values and their corresponding accounts, placing full responsibility on the operator to maintain precise ordering across multiple dimensions.

In practice, this significantly increases the risk of operator mistakes during complex batched operations, potentially causing incorrect token transfers. Furthermore, a malicious actor could deliberately misalign these arrays to execute unauthorized transfers while maintaining plausible deniability by attributing it to an ordering error.

Recommendation

We recommend replacing parallel arrays with a structured data format that explicitly binds each amount and decimal to its associated source, mint, and destination accounts.

This would eliminate reliance on positional assumptions and mitigate risks associated with misordered input.

Status: Resolved

9. Missing frozen-account check in token validation

Severity: Minor

In `contracts/anchor/withdraw/programs/withdraw/src/utils/validation.rs`, the helper that validates SPL token transfers never checks whether the source token account is frozen.

Issuers such as USDC can freeze accounts. If a frozen account is used as the from address, the transfer fails at the token-program level.

Recommendation

We recommend extending the validation to reject transfers when the token account is frozen.

Status: Resolved

10. Integer overflow in batch status tracking masks transfer failures

Severity: Minor

In `withdraw.rs:133-137` and `multi_send.rs:223-228`, the `withdraw` and `multi_send` instructions implement a batch operation status tracker using a `final_status` 64-bit integer. Each transfer's success or failure is recorded by shifting this integer left and setting the least significant bit accordingly.

However, this design is inherently incompatible with the program's `MAX_ARRAY_LENGTH` constant, which allows up to `100` transfers.

If more than `64` transfers are executed in a batch, bitwise shifts beyond the 64-bit limit cause silent truncation of the most significant bits. This truncation leads to the loss of failure records for the earliest operations, thereby masking operational errors and creating inconsistencies between the execution log and actual results.

Although Solana currently does not support execution of `100` transfers due to compute limits, this vulnerability remains latent and may be exploitable if constraints change.

Recommendation

We recommend reducing `MAX_ARRAY_LENGTH` to `64` or replacing the 64-bit integer with a more robust structure for tracking batch status, such as a dynamic bitmap or explicit array of result flags.

Status: Resolved

11. Fixed signature index prevents priority fee instructions

Severity: Informational

In `contracts/anchor/withdraw/programs/withdraw/src/instructions/multi_send.rs:114`, the code hard-codes the signature instruction index to `0`.

Because Solana priority-fee compute-budget instructions must precede all program instructions, fixing the signature at index 0 blocks callers from adding those fee instructions, forcing zero priority fees and risking slow or failed confirmations under network load.

Recommendation

We recommend deriving the signature position as `current_index - 1`, so users can insert compute-budget (priority-fee) instructions at the start of the transaction.

Status: Resolved

12. Redundant reentrancy guard reset before error return

Severity: Informational

In:

- `contracts/anchor/withdraw/programs/withdraw/src/instructions/multi_send.rs:89`,
- `contracts/anchor/withdraw/programs/withdraw/src/instructions/multi_send.rs:96`,
- `contracts/anchor/withdraw/programs/withdraw/src/instructions/multi_send.rs:147`

- `contracts/anchor/withdraw/programs/withdraw/src/instructions/withdraw.rs:52`
- `contracts/anchor/withdraw/programs/withdraw/src/instructions/withdraw.rs:74`
- `contracts/anchor/withdraw/programs/withdraw/src/instructions/withdraw.rs:121`

The handler sets reentrancy guard parameters to false and immediately afterward it returns an error, so the entire instruction is rolled back and these assignments are never persisted. The writes only consume extra compute units and make the logic harder to follow.

Recommendation

We recommend removing the guard-reset statements if the instruction errors.

Status: Resolved

13. The initialization of the program can be front-run

Severity: Informational

In `contracts/anchor/withdraw/programs/withdraw/src/instructions/initialize.rs:1-34`, the program's `initialize` function lacks access control, permitting any externally owned account to invoke it before the intended deployer.

A malicious actor could front-run the legitimate initializer, thereby claiming critical roles such as `admin`, `operator`, or `beneficiary`.

Upon successful unauthorized initialization, subsequent legitimate attempts to initialize the contract fail because all program-derived addresses are already instantiated.

This results in a permanent denial of access for the rightful owner and could compromise the system's intended governance and control mechanisms.

Recommendation

We recommend implementing strict access control on the initialize function by either:

- Requiring the transaction to be signed by the program's upgrade authority using the `Signer` pattern, or
- Enforcing a designated initializer `Pubkey`

Status: Resolved

14. Inefficient logging with `msg!` macro

Severity: Informational

In multiple instruction handlers, the program emits operational data with the `msg!` macro instead of Anchor events.

`msg!` logs consume more compute units and can be truncated or omitted by validators/RPC nodes. If logs are dropped, off-chain indexers and monitoring bots lose critical data, degrading observability and incident response.

Recommendation

We recommend replacing all `msg!` calls with the CPI-based Anchor event mechanism (`emit!`).

Status: Resolved

15. Inconsistent amount check bypasses `validate_transfer` flag

Severity: Informational

In `contracts/anchor/withdraw/programs/withdraw/src/instructions/withdraw.rs:70`, the handler verifies the transfer amount before calling `validate_token_transfer`.

When `validate_transfer` is set to `false`, callers expect the program to skip failed transfers and continue execution, but this early check still aborts the entire transaction.

Recommendation

We recommend moving the amount verification into `validate_token_transfer` (or gating it on `validate_transfer == true`) so that skipped transfers behave as intended and do not halt the overall withdrawal flow.

Status: Resolved

16. Programs should implement a two-step ownership transfer

Severity: Informational

The programs within the scope of this audit allow the current owner to execute a one-step ownership transfer.

While this is common practice, it presents a risk for the ownership of the contract to become lost if the owner transfers ownership to an incorrect address. A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and actually execute the config update.

Recommendation

We recommend implementing a two-step ownership transfer. The flow can be as follows:

1. The current owner proposes a new owner address that is validated and lowercased.
2. The new owner account claims ownership, which applies the configuration changes.

Status: Acknowledged

17. Miscellaneous comments

Severity: Informational

The codebase contains several maintenance and clarity issues:

- Uses an Anchor release older than v0.31.1, missing the latest improvements and CU optimisations.

- No `rust-toolchain.toml` pinning of Rust version. Also missing Anchor pinning of solana version, anchor version and node package manager.

- The `operation_id` field is inconsistently treated as a timestamp in `contracts/anchor/withdraw/programs/withdraw/src/instructions/withdraw.rs:45` but as a `request_id` in `contracts/anchor/withdraw/programs/withdraw/src/instructions/multi_send.rs`. It is never validated for uniqueness or format.

- Parameters named `ata` refer to arbitrary token accounts - from, beneficiary, treasury and user are not verified to be associated-token accounts.

- `contracts/anchor/withdraw/programs/withdraw/src/instructions/multi_send.rs:110` fetches `Clock::get()?.unix_timestamp` instead of reusing the already supplied timestamp, wasting compute units.

- The `status` is set to `true` in `contracts/anchor/withdraw/programs/withdraw/src/instructions/withdraw.rs:124` and `contracts/anchor/withdraw/programs/withdraw/src/instructions/multi_send.rs:213`, although it is already `true` when reaching this statement.

- The `ix_sysvar` account which is checked in `contracts/anchor/withdraw/programs/withdraw/src/state.rs:229` is verified within the `load_instruction_at_checked` function anyway.

Recommendation

We recommend:

- Upgrade the stack to the latest stable Solana and Anchor releases (≥ v0.31.1).
- Add a `rust-toolchain.toml` and `Anchor.toml` that pins specific Rust and Solana versions.
- Adopt a single meaning for `operation_id` (e.g., request ID), enforce its format, and validate uniqueness.
- Either validate that from, beneficiary, treasury and user are Associated Token Accounts or rename the parameters to reflect that no ATA guarantee is provided.
- Reuse the supplied timestamp instead of calling `Clock::get()?.unix_timestamp` to save compute units.
- Remove the superfluous setting of the `status` variable.
- Remove the redundant verification of the `ix_sysvar` account.

Status: Resolved

Appendix: Test Analysis

This analysis summarizes the current state of test coverage for the withdraw Solana program. All instruction modules, utility files (state, utils, error), and two large test suites were reviewed.

Coverage is grouped by feature for quick reference on what's tested and what's missing.

Initialization

Covered

- `initialize`: happy path & already-initialized error.
- `initialize_nonces` & `initialize_reentrancy_guard`: happy path via test setup hooks.

Missing

- Default values written by `initialize_nonces` (`current_nonce == 0`) and `initialize_reentrancy_guard` (`is_processing == false`).
- Behavior when called twice (expect "already in use" error).
- That PDA bumps returned by Anchor remain constant.

Admin instructions

Covered

- `set_admin`, `set_operator`, `set_beneficiary`: success and `NotAdmin` errors.

Missing

- Tests for `set_multi_send_operator`, `set_treasury`, `set_signer`, `set_pause` are not defined.
- Suggested table-driven tests for success/failure cases and state mutation assertions.

`withdraw` instruction

Covered

- Single & batch happy paths (SPL, SPL-2022, mixed).
- Rejections: `NotOperator`, length mismatch.

- Partial success branches: No balance, wrong delegate, insufficient allowance, ATA missing, beneficiary mismatch, token-program mismatch.
- Batch stress test (53 items).
- Return data (indirectly via assertions in test cases)

Missing

- `ZeroAmount` (early return).
- `ArrayTooLong` (`len > 100`).
- `ProgramPaused` (needs `set_pause(true)`).
- Re-entrancy guard: Test for `OperationInProgress` and `OperationTimeout`.
- `UnexpectedFromOwner` (ATA owned by wrong program).
- `InvalidTokenAccount`, `TokenAccountNotFound` (random/empty accounts).
- Decimals mismatch (provided decimals not matching mint decimals)

`multi_send` instruction

Covered

- Signature happy path.
- Partial failures: no balance, no allowance.
- Errors: `InvalidNonce`, `DeadlineExpired`, `NotOperator`, missing/invalid signature, length mismatch.
- Return data (indirectly via assertions in test cases)

Missing

- `ZeroAmount` (early return).
- `ArrayTooLong` (`len > 100`).
- `ProgramPaused`, note that `multi_send` checks a different paused flag (`MultiSendConfig.paused`).
- SPL-2022 mint(s) are unused in test cases.
- Errors: `FromOwnerMismatch`, `WrongDelegatedAddress`.
- Re-entrancy logic & clock-drift buffer edge case.

- Decimals mismatch (provided decimals not matching mint decimals)