Solido Public Report

PROJECT: Solido Audit

July 2021

Prepared For:

Solido

Chorus One AG

Prepared By:

Jonathan Haas | Bramah Systems, LLC.

jonathan@bramah.systems

Table of Contents

Solido Protocol Review

Executive Summary

Scope of Engagement

Bramah Systems, LLC was engaged in June of 2021 to perform a comprehensive security review of the Solido smart contracts (specific contracts denoted within the appendix). Our review was conducted over a period of four business days by both members of the Bramah Systems, LLC. executive staff.

Bramah Systems completed the assessment using manual, static and dynamic analysis techniques.

Timeline

Review Commencement: June 30, 2021

Report Delivery: July 5th, 2021

Engagement Goals

The primary scope of the engagement was to evaluate and establish the overall security of the Solido protocol, with a specific focus on trading actions. In specific, the engagement sought to answer the following questions:

- Is it possible for an attacker to steal or freeze tokens?
- Does the Rust code match the specification as provided?
- Is there a way to interfere with the contract mechanisms?
- Are the arithmetic calculations trustworthy?

Contract Specification

Specification was provided in the form of code comments. The contracts were provided via GitHub (commit hash **e714112f58109780e469c833db2fc9b7fb5f19f6**)

Overall Assessment

Bramah Systems was engaged to evaluate and identify any potential security concerns within the codebase of the Solido protocol. During the course of our engagement, Bramah Systems found minimal instances wherein the team deviated materially from established best practices and procedures of secure software development within DLT. The team has since addressed these issues with a resolution or risk acceptance.

Disclaimer

Timeliness of Content

General Recommendations

Best Practices & Rust Development Guidelines

Instances of unwrap can be made more user friendly

Because the **unwrap** function may panic, its use is generally discouraged. Instead, we suggest using pattern matching and handling the **Err** case explicitly, or calling **unwrap_or**, **unwrap_or_else**, or **unwrap_or_default**. Consider where possible replacing the unwrap with a more user-friendly error. For example, in the instance of **withdraw_active_stake** and **collect_validator_fee** (amongst others in **instruction.rs**) in which a code comment makes apparent there is no anticipated reason for failure, perhaps an **expect** would be more appropriately used.

This noted, there are certain instances wherein usage of unwrap should not present concern, including but not limited to tests and diagnostic functions explicitly intended for unwrap usage in which a more viable alternative was not determined to be present.

Resolution:

The Lido team provided the following details:
In **program/:**

- Use of unwrap in tests/ and in tests mod is fine and intentional.
- All cases in accounts.rs are in tests.
- All uses in instruction.rs are due to a bad type signature of BorshSerialize::to_vec. This function returns a Result because it calls BorshSerialize::serialize, which works with an arbitrary writer, which can cause an IO error, and therefore returns Result. But when serializing to a vector, there is no IO, and therefore to_vec should never fail. There is a comment next to these uses of .unwrap() that explains this, but I suppose we could make it more explicit by using .expect() instead.
- In state.rs, one occurrence is due to Borsh again, and the others are all in tests.

The all occurrences of .unwrap(in **cli/:**

- In config.rs we use unwrap because the config can be in a partially validated state early in the program. If you call these functions that call .unwrap() earlier, this is a programming error, and it should panic. There are ways to make the invalid state unrepresentable, but I don't think those are worth the trouble at this point.

- In daemon.rs, two unwraps are due to Rust's lock poisioning design mistake. It does not affect us, because we set panic = "abort", so a panic aborts the program rather than poisioning the lock. One unwrap is due to thread joining, which again does not affect us because we set panic = "abort".
- One unwrap is a true case of an error that would be nice to handle: if we fail to start the http server, we can print a nice error message. I'll address this.
- All unwraps in maintenance.rs are in tests.
- All unwraps in prometheus.rs are in tests.

Bramah has confirmed each test element as represented, and confirmed unwrap's usage as satisfactory. Bramah has also confirmed the modification to the HTTP server error message, and confirmed its usage as satisfactory as well.

Specific Recommendations

Unique to the Solido Protocol

Improper validation of Account writable and signer properties in try_from_slice in accounts_struct!

The accounts_struct macro makes the construction of account Meta and Info-suffixed structs easier. A try_from_slice implementation is generated at compile-time for both Meta and Info struct idents. It attempts to take a slice of accounts and create the respective struct after performing some validation that the provided accounts appear in-order and match the expected properties as defined by the macro-caller through is_signer and is_writable properties.

There is a small problem in the implementation of both try_from_slice functions as they fail to detect accounts that are marked as either being a signer or writable when the expected value for these properties is that they are not due to short-circuiting behavior in the boolean logic.

For example, if an AccountMeta was evaluated against the manager account defined under InitializeAccountsInfo it could be marked as writable despite the required property being false.

```
/*
  $is_writable => false
  $var_account.is_writable => could be true or false
*/
if $is_writable && !$var_account.is_writable {
  return Err(LidoError::InvalidAccountInfo.into()); // not reached in above scenario
}
```

Strengthening this check could ensure additional redundancy that accounts are not unnecessarily writable. Additionally, if accounts are required to be read-only here it helps improve parallelization of the code.

Resolution:

Uninitialized Manager Account is on the ed25519 Curve

The manager account under the Lido object falls on the ed25519 by default, before initialization. The lido.check_manager(...) guard is often used throughout the code to ensure that privileged operations are signed by the manager. Because the default, uninitialized manager account falls on ed25519 it's possible for someone within the Solana network to acquire the account with a public key of [0u8; 32] allowing them to sign privileged messages against an uninitialized system.

While this is an unlikely attack vector, it's not one that the Solana ecosystem avoids addressing. For example, Solana Program Addresses explicitly are generated *off* the curve so being able to sign messages with them is impossible.

> *A Program address does not lie on the ed25519 curve and therefore has no valid private key associated with it, and thus generating a signature for it is impossible. - Solana "Calling Between Programs"*

It is recommended to update the manager check to also ensure that the system has been initialized at least once (or that the manager has not been set to all 0's) in addition to the existing check.

Resolution:

Use *Info In Place of accounts_raw for Processor Functions

While no instance currently exists where the incorrect *Info struct is used in a given processor function, we did notice that implementation of the codebase was not finished and could introduce issues for code going forward if new code used the incorrect *Info struct to gate on account requirements.

As a general practice, we suggest enforcing as many requirements upfront as is feasible especially in this case where we know both the intended operation and the required number of and state of the accounts that can be passed in by a caller.

As an example:

```
pub fn process_deposit(
    program_id: &Pubkey,
    amount: Lamports,
    accounts_raw: &[AccountInfo],
) -> ProgramResult {
    let accounts = DepositAccountsInfo::try_from_slice(accounts_raw)?;

    //...
}
```

to

```
pub fn process_deposit(
    program_id: &Pubkey,
    amount: Lamports,
    accounts: DepositAccountsInfo,
) -> ProgramResult {
    //...
}
```

Callers would then perform the construction of the `*Info` type before invoking the associated processor function. The previous design allows processor functions to construct arbitrary raw_accounts using the incorrect required accounts and account permissions (signing and mutability).

Resolution:

TODO Items remain in source

Multiple TODO items remain in the source code, including certain key functionality that appears to impact users directly. This includes external notification of a TODO item (via system output) within the management CLI utility, which should be moved to a code-comment. It is suggested that these items be made clearly delineated as production and non-production blocking

Resolution:

Typographic errors in code comments

We heavily recommend a spelling linter be used upon the various code comments within the protocol. Instances include:

- “Foward” in **solido/cli/src/config.rs**

Resolution:

Out of date dependencies

The **borsh** dependency is out of date and should be updated in order to benefit from more recent updates to the software.

Resolution:

Unclear reference to Goat Teleportation in Prometheus

There appears to be a seemingly random reference to goat teleportation within the CLI utility. It’s unclear if this purpose is in jest or is used to test the prometheus configuration. Either way, content not core to the operating purpose of the system should be removed, or have its purpose clearly indicated.

Resolution:

Clear distinction between management utilities and user-facing components should be made

The management utility references an element which relays the following:

```
“// Unfortunately, there is no way to get a structured error; all we
        // get is a string that looks like this:
        //
        //     Failed to deserialize RPC error response: {"code":-32602,
        //     "message":"Too many inputs provided; max 100"} [missing field `data`]
        //
```

// So we have to resort to testing for a substring, and if Solana

// ever changes their responses, this will break :/”

While normally this could present concern (as such behaviour has happened to other blockchain platforms to user-facing components), this code exists primarily within the management utility, lowering concerns. However, it should be made clear to users that this is the case, and that the management utility is not intended for public usage.

Resolution:

Expect can be rewritten for clarity

There is a reference to the following code in **stake_account.rs** that can be rewritten for clarity, as the present reference to activity and activation in many states may be confusing to some:

```
let inactive_lamports = account_lamports.0
        .checked_sub(active_lamports)
        .expect("Active stake cannot be larger than stake account balance.")
        .checked_sub(activating_lamports)
        .expect("Activating stake cannot be larger than stake account balance - active.")
        .checked_sub(deactivating_lamports)
        .expect("Deactivating stake cannot be larger than stake account balance - active -
activating.");
```

Resolution:

Toolset Warnings

Unique to the Solido Protocol

Overview

In addition to our manual review, our process involves utilizing static analysis in order to perform additional verification of the presence of security vulnerabilities (or lack thereof). An additional part of this review phase consists of reviewing any automated unit testing frameworks that exist.

The following sections detail warnings generated by the automated tools and confirmation of false positives where applicable.

Compilation Warnings

No compilation warnings were encountered during the course of our audit.

Test Coverage

The contracts possess a number of functional unit tests encompassing various stages of the application lifecycle.

Static Analysis Coverage

The contract repository underwent scrutiny with static analysis agents, but largely benefits from the inherent security posture that Rust provides. Rust statically enforces many properties of a program beyond memory safety, including null pointer safety and data race safety. With a well defined test suite and ample documentation as to the functionality, the Solido contracts go into deep detail as to how each implementation functions.

Directory Structure

At time of review, the directory structure of the Solido smart contracts repository appeared as it does below. Our review, at request of Solido, covers the Rust code (*.rs) as of commit hash **e714112f58109780e469c833db2fc9b7fb5f19f6**

```
.
├── CHANGELOG.md
├── Cargo.lock
├── Cargo.toml
├── README.md
├── buildimage.sh
├── cli
│   ├── Cargo.toml
│   └── src
│       ├── config.rs
│       ├── daemon.rs
│       ├── error.rs
│       ├── helpers.rs
│       ├── main.rs
│       ├── maintenance.rs
│       ├── multisig.rs
│       ├── prometheus.rs
│       ├── spl_token_utils.rs
│       ├── stake_account.rs
│       └── util.rs
├── docker
│   ├── Dockerfile.base
│   ├── Dockerfile.dev
│   ├── Dockerfile.maintainer
│   ├── docker-compose.yml
│   └── entrypoint.sh
├── multisig
├── program
│   ├── Cargo.toml
│   ├── Xargo.toml
│   ├── src
│   │   ├── account_map.rs
│   │   ├── accounts.rs
│   │   ├── balance.rs
│   │   ├── entrypoint.rs
│   │   ├── error.rs
│   │   ├── instruction.rs
│   │   ├── lib.rs
│   │   ├── logic.rs
│   │   ├── process_management.rs
│   │   ├── processor.rs
│   │   ├── state.rs
│   │   ├── token.rs
│   │   └── util.rs
│   └── tests
│       ├── context.rs
│       ├── mod.rs
│       └── tests
│           ├── add_remove_validator.rs
│           ├── change_reward_distribution.rs
│           ├── deposit.rs
│           ├── maintainers.rs
│           ├── merge_stake.rs
│       ├── mod.rs
│       ├── stake_deposit.rs
│       ├── update_exchange_rate.rs
│       └── update_validator_balance.rs
├── solana-program-library
└── tests
    ├── README.md
    ├── prepare_test_environment.py
    ├── start_test_validator.py
    ├── test_multisig.py
    ├── test_solido.py
    └── util.py

10 directories, 54 files
```