ackee blockchain

Marinade Audit

1. 9. 2021

by Ackee Blockchain

Table of Contents

1. Overview

1.1 Ackee Blockchain

Ackee Blockchain is an auditing company based in Prague, Czech Republic, specialized in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run a free certification course Summer School of Solidity and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, Rockaway Blockchain Fund.

1.2 Audit Methodology

1. **Technical specification/documentation** - a brief overview of the system is requested from the client and the scope of the audit is defined.
2. **Manual code review** - the code is checked line by line for common vulnerabilities, code duplication, best practices and the code architecture is reviewed.
3. **Local deployment + hacking** - the program is deployed locally and we try to attack the system and break it.

1.3 Review team

The audit has been performed with a total time donation of 1 engineering month. The work was divided between the Chief Auditor and two auditors who performed manual code review. The whole process was supervised by the Audit Guarantee.

Member's Name	Position
Stepan Sonsky	Chief Auditor

Auditor 1	Ackee developer
Auditor 2	External developer
Josef Gattermayer, Ph.D.	Audit Guarantee

1.4 Disclaimer

We've put our best effort to find all vulnerabilities in the system, however our findings shouldn't be considered as a complete list of all existing issues.

2. Scope

This chapter describes the audit scope, contains provided specification, used documentation and set main objectives for the audit process.

2.1 Coverage

Files being audited:

- /programs/marinade-finance

Sources revision used during the whole auditing process:

- Repository: **https://github.com/marinade-finance/marinade-anchor**
- Commit:0fdd2f0d641dc09ce2d9adc74dbab9d8010c5a09

2.2 Supporting Documentation

Most of the documentation is located in the Backend-Design.md file located in the /doc/ directory in the main repository. This file contains information about the overall design, the basic and advanced interface, the crank bot and also a section with user stories and happy paths which were added after our inquiry three weeks into the audit.

Other documentation is available in the form of in-code comments (which deal with more technical concepts and issues) and sporadic code annotations.

2.3 Objectives

We've defined following main objectives of the audit:

- Check the overall code quality.
- Make sure that nobody unauthorized can withdraw SOL or mSOL from the liquid pool.
- Verify that only Marinade itself can mint tokens.
- Check that only authorized entities can deploy the program to the Solana network.

3. System Overview

This chapter describes the audited system from our understanding.

The basic idea behind Marinade is to allow liquid SOL staking and un-staking, because there's no cooldown period for the transactions and the user receives tokens immediately (for the Basic interface) or after a pre-defined number of epochs (for the Advanced interface) and doesn't have to wait for several days. Marinade also provides a way for users to earn fees after depositing their SOL into the liquid pool.

3.1 Key components

Liquid pool: which holds staked SOL and mSOL.

Reserve account: an intermediate storage for SOL for balancing stakes/un-stakes. SOLs are transferred from the reserve account when the user claims a valid (unstake) ticket.

Stake system: that implements the logic for delay unstaking. Deals with token unstaking itself, stake accounts depositing, stake accounts merging and deactivating.

Crank Bot: that processes stake accounts (stakes, unstakes, adds rewards), recalculates mSOL price, removes deactivated accounts and recalculates mSOL price if rewards were received. Action taken by the bot depends on the delta between stake and unstake orders.

Interfaces: the marinade-finance program contains two interfaces - Basic and Advanced. These are described in the next section.

3.2 Basic interface

3.2.1 Liquid stake

Deposit process depends on how many SOL is the user depositing and how many mSOL are there in the liquid pool. In the basic scenario when the liquid pool is fully unbalanced (ie. it contains only SOL and no mSOL): Marinade receives the user's SOL. SOL is then deposited in the reserve account (and are later staked into the liquid pool if they're not used for liquid unstaking) and appropriate amount of mSOL is minted for the user (if there aren't enough mSOL in the liquid pool). The amount is calculated as a number of deposited SOL * share price, where share price is staked SOL / total mSOL.

In case there is an unstake operation happening simultaneously the user can receive mSOL sent by the other user directly and no additional tokens have to be minted provided the staked amount of mSOL is less than the unstaked.

3.2.2 Liquid unstake:

Unstaking begins with a user sending mSOL. Appropriate amount of SOL is then removed from the liquid pool and sent to the user. The amount of returned SOL depends on how much is the user unstaking. If everything - the maximum fee is applied which is initially set as 3%. Otherwise it's calculated based on the remaining amount in the liquid pool and target liquid pool liquidity and ranges between maximum and minimum fee..

3.3 Advanced interface

3.3.1 Deposit-Stake-Account

Users can deposit only a delegated active account which is not in the cooldown period. Marinade takes over the account by becoming its Stake and Withdrawal authority then the user receives an appropriate amount of mSOL sans fees.

3.3.2 Delayed Unstake

The delayed unstake process is initiated when a user requests to unstake SOL. Received mSOL are burned, an unstake order is registered and an unstake ticket account is created for the user. Current epoch is stored in this account and it enters a cool down phase worth several epochs (currently set to 2). After the cool down period the user can withdraw SOL (claim the ticket) and the account is eventually deleted by the crank-bot. Users can start several delayed unstakes and receive ticket accounts for each of them.

Depending on the delta between depositing and unstaking two scenarios can occur.

There are more SOL unstaked then deposited. In this case when the Bot runs at the end of the epoch the staked SOL are reserved for future unstaking. The bot also starts the unstaking process for the additional SOL. At the end of the cooldown period the bot receives the requested amount of additional SOL and the reserve account now holds enough SOL so the user receives them.

If the delta is positive the user receives SOL and delta SOL are sent to the liquid pool. All stake rewards are staked.

4.Security Specification

This section specifies single roles and their relationships in terms of security in our understanding of the audited system. These understandings are later validated.

4.1 Actors

This part describes actors of the system, their roles and permissions.

Owner + Liquidity Provider

The owner deploys the Marinade program to the Solana network. Marinade then transfers funds between the liquid pool, the reserve account and sends a cut of fees to the treasury account.

User

Deposits SOL tokens and stake accounts into Marinade and receives mSOL tokens minted by Mariande in return. Users can also withdraw SOL tokens in exchange for mSOL.

4.2 Trust model

Users have to trust developers that math algorithms are correctly implemented and they receive the correct amount of rewards (if any) and mSOL for SOL and vice-versa.

5. Findings

This chapter shows detailed output of our analysis and testing.

5.1 General Comments

This section presents an overall engineering culture that is a crucial precursor of the right security.

5.1.1 Overall code quality

There are numerous instances of commented out code which is a sign of an unstable code not ready for production. Multiple occurrences of additional sanity checks are commented out without any comments explaining why those checks aren't needed. For some examples see Appendix E. These need to be removed from the repository or documented.

The code should follow Rust style guide and best practices recommendations. Tools like cargo-clippy should be used (see Appendix B).

5.1.2 Commit culture

Considerable portions of the commit messages are generic and don't go into any detail about the code that's being commited. There are numerous "fix a bug" commit messages that don't explain what the bug was and how it was mitigated. This makes auditing or even verifying the changes by other developers difficult. Some examples:

> commit 6c13e25b "Fixes"
> commit 304f9d2c "fix emergency unstake"
> commit c2abe719 "FIx list remove bug"
> commit 268a2b39 "fix and enhance bot after first mainnet run"
> commit b7771503 "fix"

Unless the changes to the code are truly self explanatory a proper commit message should include description of the change set.

Adopting a standardized commit message format which would contain the name of the component (tests, crank-bot, liquid pool etc.) a short description and a more detailed explanation would make future auditing and code review easier.

We also strongly suggest making Pull Requests and peer code review before committing mandatory for each commit.

5.1.3 Comments and documentation

There is sufficient documentation on how the program is supposed to work in general and the relations between the various components. However there is much more to be covered. How does the validators selection works, how exactly are the fees and rewards calculated and there is no documentation regarding error handling and recovery.

Although critical parts of the code are sufficiently commented there are many more which lack comments completely. We suggest mandatory rustdoc annotations for all methods so documentation can be generated automatically.

5.1.4 Release cycle

There is no clear roadmap towards a stable release ie. which features should be finished and included into each deployment. There are no tags in the git repository, which would mark a specific commit in the repository as a release candidate. The proper release cycle starts with tagging a specific commit, building that revision and running tests against it. Only after all tests pass can the release candidate be deployed to the network. We didn't find anything that would suggest this release strategy is used.

5.1.5 Logging

There is extensive logging in the most critical parts of the code but most of these log messages have arbitrary format that makes them impossible to automatically parse. A rigid log message format would enable automatic processing of the debug log output and automatic notifications could be sent upon detecting suspicious activity. An example of such a message is in `src/liq_pool/remove_liquidity.rs` on line 93
"Someone minted lp tokens without our permission or bug found".
Log entries like this are impossible to spot with only human supervision. We recommend using a log crate to log status (warn, info, fatal, etc.), message and additional data for easier debugging, setting the alerts or machine processing.

5.2 Issues

Using our toolset, manual code review we've identified the following issues.

Low

Low severity issues are more comments and recommendations rather than security issues. We provide hints on how to improve code readability and follow best practices. Further actions depend on the development team decision.

ID	Description	Contract	Line
L1	Not using a stable toolchain	programs/marinade-finance/src/lib.rs	1
L2	Not using a linter tool	programs/marinade-finance	
L3	Using the outdated dependencies	programs/marinade-finance	
L4	Repository contains deploy keys	keys/marinade_finance-keypair.json	

L1: Marinade program is not possible to build with a fully stable Rust environment. Currently the program can be build only by the nightly version of Rust. Development must be done using a fully stable toolchain, for limiting potential compiler, runtime or tool bugs. There is a problem with #![feature(proc_macro_hygiene)] that is not allowed for the stable toolchain.

L2: Marinade program contains code warnings (like unnecessary reference, etc.) that should be fixed. A linter must be used regularly during the development of a secure application. There should be a lint check added as a new step in your build pipeline or pre-commit hook. The clippy linter could be used (cargo clippy) to check and fix some of the found warnings automatically by running a `cargo clippy` or `cargo clippy --fix` command.

The result of the cargo-clippy command could be seen in Appendix B

L3: Marinade program uses outdated dependencies. Each outdated dependency must be updated or the choice of the version must be justified. The `cargo-outdated` or `cargo-upgrades` tool must be used to check dependencies status. There should be an outdated check as a new step in your build pipeline.

The result of the cargo-upgrades command could be seen in Appendix C

L4: Marinade repository contains deploy keys that could cause security problems. With attached deploy keys developers are able to deploy a new program version to the testnet, devnet. These keys should take place in some vault or special storage for application credentials. Having deploy keys in a repository is a bad practice.

Medium

Medium severity issues aren't security vulnerabilities, but should be clearly clarified or fixed.

ID	Description	Contract	Line
M1	Using deprecated libraries	programs/marinade-finance	

M1: Marinade program uses libraries that are deprecated. These libraries could potentially cause security vulnerabilities. The `cargo-audit` tool must be used to check for known vulnerabilities in dependencies. There should be a cargo-audit check in your build pipeline to make sure you are using secure dependencies.

The result of the cargo-audit command could be seen in Appendix D

High

High severity issues are security vulnerabilities, which require specific steps and conditions to be exploited. These issues have to be fixed.

ID	Description	Contract	Line
H1	Using unaudited Anchor framework	programs/marinade-finance	

H1: Marinade relies on the Anchor framework for the Solana Sealevel runtime. Anchor developers state in their official documentation that:

- Anchor is in active development, so all APIs are subject to change.
- This code is unaudited. Use at your own risk.

Production code should not rely on unstable, unaudited and thus insecure code.

Critical

Direct critical security threats, which could be instantly misused to attack the system. These issues have to be fixed.

✓ We haven't found any critical severity issues.

5.3 Testing & Verification

The project implements two types of tests: Unit Tests, Integration Tests.

- Unit Tests
  - The project has modest and incomplete unit tests coverage.
  - Unit test code coverage is not sufficient for production release.
  - Branch coverage is 1.21%
  - Function coverage is 1.67%
  - Lines coverage is 2.74%
- Integration Tests
  - The project has extensive Integration tests coverage.
  - The project tests are not completed - there are some todos and comments, for example:
    - delayed_unstake.rs - lines 79, 93, 186
    - delayed_unstake.rs - line 89
    - test_add_remove_liquidity.rs - lines 94, 110, 136, 143
    - test_add_remove_liquidity.rs - lines 119 - 327

    Production code should not contain todos or commented functions, methods.

General code coverage is not sufficient for production release. We recommend extending unit test coverage by adding more tests and covering edge cases.

For Unit Test Code Coverage see Appendix A.

Notes

For failed tests see Appendix B.

6. Conclusion

The first auditing week was dedicated to general understanding of the whole system. We discovered here first general issues - low level of developer documentation and bad commit culture. Due to these issues we had to extend the general understanding period to the second week as the system was still not fully understood by us.

We followed in the third week with a top-down manner starting from overall code quality and ending with the actual code logic and possible exploits. Our progress was hindered by missing code annotations and missing happy path user scenarios.

The code shows clear signs of rapid development where speed stakes precedence over best coding practices. Combined with relying on other software that is still under development makes introducing bugs into the code base inevitable.

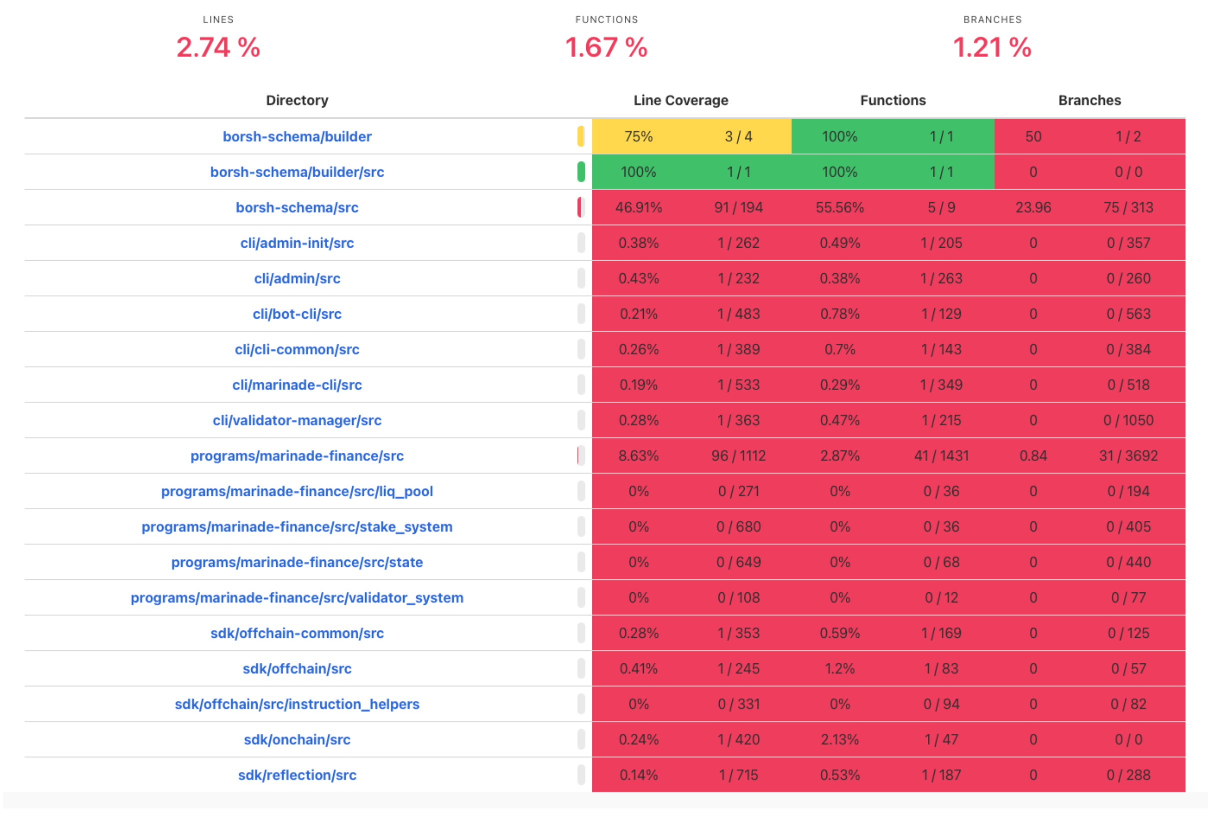
The test coverage of unit tests is very poor (~2%), integration tests cover more of the project but there are still a lot of missing tests.

We have discovered a medium severity issue and a few more with low severity. None of the issues requires immediate action.

We believe the project lacks technical leadership with clear rules and guidelines for development, commit messages, log messages, coding style, comments and documentation, peer reviews between developers and a clear roadmap for features and deployment. This would definitely help future auditors (or developers) better understand the code and be able to focus on finding single issues.

A next more in depth audit should follow once the internal technological debt is eliminated.

Appendix A



LINES 2.74 %
FUNCTIONS 1.67 %
BRANCHES 1.21 %

Directory		Line Coverage		Functions		Branches	
borsh-schema/builder	🟨	75%	3 / 4	100%	1 / 1	50	1 / 2
borsh-schema/builder/src	🟩	100%	1 / 1	100%	1 / 1	0	0 / 0
borsh-schema/src	🟥	46.91%	91 / 194	55.56%	5 / 9	23.96	75 / 313
cli/admin-init/src		0.38%	1 / 262	0.49%	1 / 205	0	0 / 357
cli/admin/src		0.43%	1 / 232	0.38%	1 / 263	0	0 / 260
cli/bot-cli/src		0.21%	1 / 483	0.78%	1 / 129	0	0 / 563
cli/cli-common/src		0.26%	1 / 389	0.7%	1 / 143	0	0 / 384
cli/marinade-cli/src		0.19%	1 / 533	0.29%	1 / 349	0	0 / 518
cli/validator-manager/src		0.28%	1 / 363	0.47%	1 / 215	0	0 / 1050
programs/marinade-finance/src	🟥	8.63%	96 / 1112	2.87%	41 / 1431	0.84	31 / 3692
programs/marinade-finance/src/liq_pool		0%	0 / 271	0%	0 / 36	0	0 / 194
programs/marinade-finance/src/stake_system		0%	0 / 680	0%	0 / 36	0	0 / 405
programs/marinade-finance/src/state		0%	0 / 649	0%	0 / 68	0	0 / 440
programs/marinade-finance/src/validator_system		0%	0 / 108	0%	0 / 12	0	0 / 77
sdk/offchain-common/src		0.28%	1 / 353	0.59%	1 / 169	0	0 / 125
sdk/offchain/src		0.41%	1 / 245	1.2%	1 / 83	0	0 / 57
sdk/offchain/src/instruction_helpers		0%	0 / 331	0%	0 / 94	0	0 / 82
sdk/onchain/src		0.24%	1 / 420	2.13%	1 / 47	0	0 / 0
sdk/reflection/src		0.14%	1 / 715	0.53%	1 / 187	0	0 / 288

Appendix B

The result of the cargo-clippy command.

```
...
Checking marinade-finance v0.1.0 (./marinade-anchor/programs/marinade-finance)

warning: this expression borrows a reference (`&anchor_lang::prelude::Pubkey`) that is
immediately dereferenced by the compiler
--> programs/marinade-finance/src/stake_system/merge.rs:42:13
| 42
|             &self.stake_program.key,
|             ^^^^^^^^^^^^^^^^^^^^^^^ help: change this to: `self.stake_program.key`
|
= note: `#[warn(clippy::needless_borrow)]` on by default
= help: for further information visit
https://rust-lang.github.io/rust-clippy/master/index.html#needless_borrow

warning: this lifetime isn't used in the function definition
--> programs/marinade-finance/src/state.rs:237:30
| 237
|     fn check_reserve_address<'info>(&self, reserve: &Pubkey) -> ProgramResult;
|                              ^^^^^
|
= note: `#[warn(clippy::extra_unused_lifetimes)]` on by default
= help: for further information visit
https://rust-lang.github.io/rust-clippy/master/index.html#extra_unused_lifetimes

warning: this expression borrows a reference (`&anchor_lang::prelude::Pubkey`) that is
immediately dereferenced by the compiler
--> programs/marinade-finance/src/validator_system/remove.rs:25:52
| 25
|             != &validator.duplication_flag_address(&self.state.to_account_info().key)
|                                                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
help: change this to: `self.state.to_account_info().key`
|
= help: for further information visit
https://rust-lang.github.io/rust-clippy/master/index.html#needless_borrow

warning: this expression borrows a reference (`&anchor_lang::prelude::Pubkey`) that is
immediately dereferenced by the compiler
--> programs/marinade-finance/src/validator_system/remove.rs:30:52
| 30
|                 validator.duplication_flag_address(&self.state.to_account_info().key)
|                                                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
help: change this to: `self.state.to_account_info().key`
|
= help: for further information visit
https://rust-lang.github.io/rust-clippy/master/index.html#needless_borrow

warning: `marinade-finance` (lib) generated 4 warnings
```

```
...
```

Appendix C

The result of the cargo-upgrades command.

```
marinade-borsh-schema: ./marinade-anchor/borsh-schema/Cargo.toml
anchor-lang matches 0.11.1;     latest is 0.13.2

marinade-finance: ./marinade-anchor/programs/marinade-finance/Cargo.toml
anchor-lang matches 0.11.1;     latest is 0.13.2
anchor-spl matches 0.11.1;      latest is 0.13.2

marinade-finance-onchain-sdk: ./marinade-anchor/sdk/onchain/Cargo.toml
anchor-lang matches 0.11.1;     latest is 0.13.2

marinade-reflection: ./marinade-anchor/sdk/reflection/Cargo.toml
anchor-lang matches 0.11.1;     latest is 0.13.2
rand matches 0.7.3;     latest is 0.8.4

cli-common: ./marinade-anchor/cli/cli-common/Cargo.toml
anchor-lang matches 0.11.1;     latest is 0.13.2

mardmin-init: ./marinade-anchor/cli/admin-init/Cargo.toml
anchor-lang matches 0.11.1;     latest is 0.13.2

./marinade-anchor/cli/admin/Cargo.toml
anchor-lang matches 0.11.1;     latest is 0.13.2

validator-manager: ./marinade-anchor/cli/validator-manager/Cargo.toml
anchor-lang matches 0.11.1;     latest is 0.13.2

marinade: ./marinade-anchor/cli/marinade-cli/Cargo.toml
anchor-lang matches 0.11.1;     latest is 0.13.2
anchor-spl matches 0.11.1;      latest is 0.13.2
bs58 matches 0.3.1;     latest is 0.4.0

./marinade-anchor/cli/bot-cli/Cargo.toml
anchor-lang matches 0.11.1;     latest is 0.13.2
anchor-spl matches 0.11.1;      latest is 0.13.2
```

Appendix D

The result of the cargo-audit command.

```
...
Cargo.lock for vulnerabilities (454 crate dependencies)
Crate:         tar
Version:       0.4.35
Title:         Links in archive can create arbitrary directories
Date:          2021-07-19
ID:            RUSTSEC-2021-0080
URL:           https://rustsec.org/advisories/RUSTSEC-2021-0080
Solution:      Upgrade to >=0.4.36
Dependency tree: …

Crate:         failure
Version:       0.1.8
Warning:       unmaintained
Title:         failure is officially deprecated/unmaintained
Date:          2020-05-02
ID:            RUSTSEC-2020-0036
URL:           https://rustsec.org/advisories/RUSTSEC-2020-0036

Dependency tree: …

Crate:         net2
Version:       0.2.37
Warning:       unmaintained
Title:         `net2` crate has been deprecated; use `socket2` instead
Date:          2020-05-01
ID:            RUSTSEC-2020-0016
URL:           https://rustsec.org/advisories/RUSTSEC-2020-0016

Dependency tree: …

Crate:         bytemuck
Version:       1.7.0
Warning:       yanked

Dependency tree: …

Crate:         crossbeam-deque
Version:       0.7.3
Warning:       yanked

Dependency tree: …

Crate:         crossbeam-deque
Version:       0.8.0
Warning:       yanked

error: 1 vulnerability found!
warning: 5 allowed warnings found
```

Appendix E

Examples of commented out code without explanation on why it was disabled.

```
programs\marinade-finance\src\liq_pool\add_liquidity.rs:38

    // self.state
    //  .check_st_sol_mint(self.st_sol_mint.to_account_info().key)?;

programs\marinade-finance\src\liq_pool\remove_liquidity.rs:93

    msg!("Someone minted lp tokens without our permission or bug found");
    // return Err(ProgramError::InvalidAccountData);

programs\marinade-finance\src\state\order_unstake.rs:

        /*
        //self.ticket_beneficiary (also ticket beneficiary) must be native SOL account
        check_owner_program(
        &self.ticket_beneficiary,
        &system_program::ID,
        "ticket_beneficiary",
        )?;

        if self.burn_st_sol_from.owner != *self.ticket_beneficiary.key {
        msg!(
                "burn_st_sol_from.owner {} must be ticket_beneficiary {}",
                self.burn_st_sol_from.owner,
                *self.ticket_beneficiary.key
        );
        return Err(ProgramError::InvalidAccountData);
        }*/
```

Thank You

Ackee Blockchain a.s.

Prague, Czech Republic

hello@ackeeblockchain.com

https://discord.gg/fc6CRw9n