# Secure Code Assessment of the Allbridge EVM Contract, Solana Contract, & Info Server

Findings and Recommendations Report Presented to:

## APY Foundation, Inc

May 18, 2022

Version: 2.0

Presented by:

Kudelski Security, Inc.
5090 North 40th Street, Suite 450
Phoenix, Arizona 85018

# TABLE OF CONTENTS

# LIST OF FIGURES



# LIST OF TABLES

# EXECUTIVE SUMMARY

## Overview

APY Foundation, Inc engaged Kudelski Security to perform a Secure Code Assessment of the Allbridge EVM Contract, Solana Contract, & Info Server.

The assessment was conducted remotely by the Kudelski Security Team. Testing took place on February 02 - March 29, 2022, and focused on the following objectives:

- Provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the result of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the Kudelski Security Teams took to identify and validate each issue, as well as any applicable recommendations for remediation.

## Key Findings

The following issues were identified during the testing period. These should be prioritized for remediation to reduce the risk they pose:

- KS-AB-01 – Locks and Unlocks can be denied
- KS-AB-02 – New validators can replay previous validators' unlocks
- KS-AB-03 – Account data are not zeroized during `remove_token`
- KS-AB-04 – Comment instead of code logic
- KS-AB-05 – Loss of precision when casting during staking
- KS-AB-06 – Missing zero address validation
- KS-AB-07 – NEAR contract address is not forbidden
- KS-AB-08 – Price caching time is higher than the update time
- KS-AB-09 – Reentrancy attacks
- KS-AB-10 – Unlock may be created for unsupported token
- KS-AB-11 – Vulnerabilities in transitive dependencies
- KS-AB-12 – Vulnerable dependency
- KS-AB-13 – Different roles to start and stop the bridge
- KS-AB-14 – Extensive use of `find_program_address`
- KS-AB-15 – Inconsistency between comment and code
- KS-AB-16 – Missing Ownership Checks
- KS-AB-17 – Missing tests of behavior after reward update
- KS-AB-18 – Public visibility is set for functions that are not called internally
- KS-AB-19 – Vulnerable OpenZeppelin version

During the test, the following positive observations were noted regarding the scope of the engagement:

- The team was very supportive and open to discussing the design choices made

Based on formal verification, we conclude that the reviewed code implements the documented functionality.

## Scope and Rules of Engagement

Kudelski performed a Secure Code Assessment of the Allbridge EVM Contract, Solana Contract, & Info Server. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a private repository at https://github.com/allbridge-io/bridge-evm-contract with the commit hash 31e1c309e7d92b70482f741c60253bfd3f0d2a53, https://github.com/allbridge-io/bridge-solana-contract with the commit hash 42c94b6840410acd1eeb77d99c8b83b05df30434, and https://github.com/allbridge-io/bridge-info-server with the commit hash 822e37a2f06ebfa552cfcb6d1976011ea38a477e.

A re-review was performed on May 6, 2022, with the commit hash b9144adeb39d44e8bd564c8a9162c506eb8ef293.

# TECHNICAL ANALYSIS & FINDINGS

During the Secure Code Assessment of the AllBridge EVM Contract, Solana Contract, & Info Server, we discovered:

- 2 findings with MEDIUM severity rating.
- 10 findings with LOW severity rating.
- 7 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

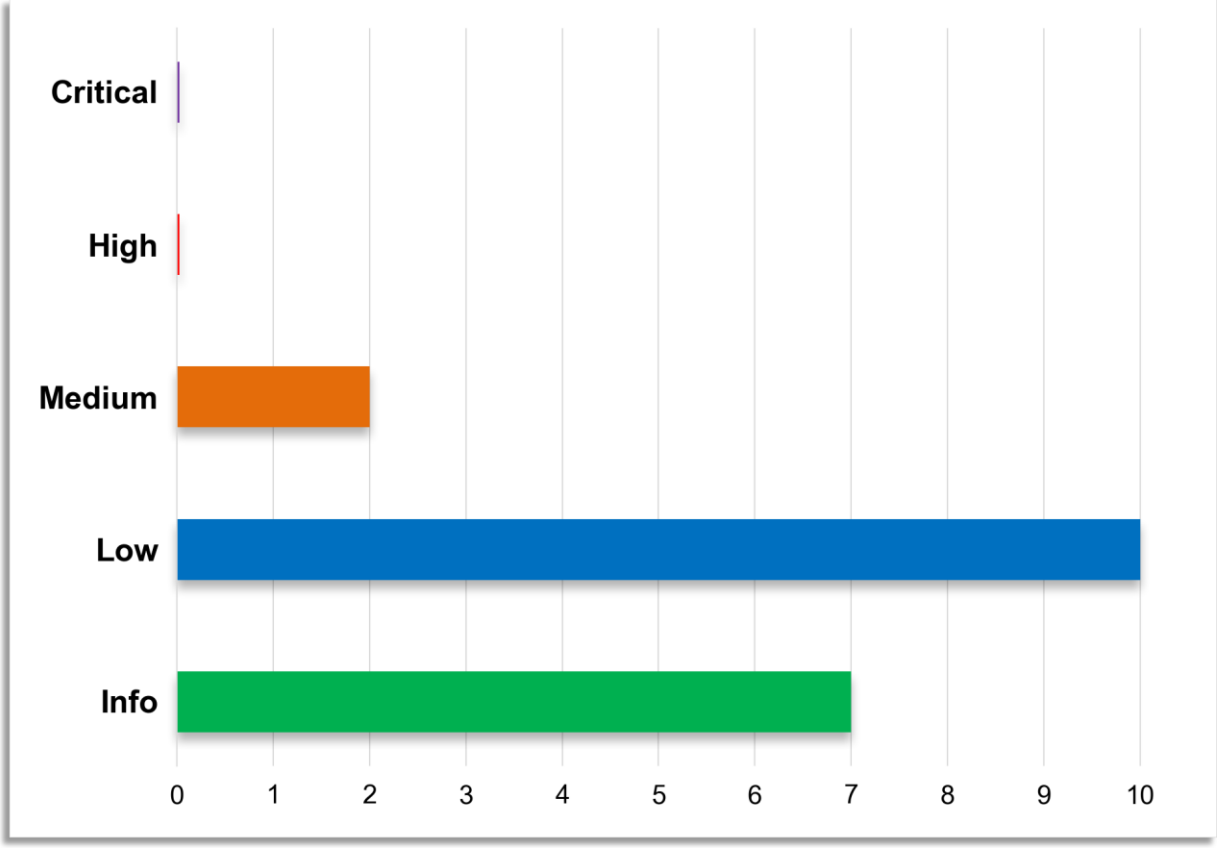


Figure 1: Findings by Severity

## Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

| # | Severity | Description |
|---|---|---|
| KS-AB-01 | Medium | Locks and Unlocks can be denied |
| KS-AB-02 | Medium | New validators can replay previous validators' unlocks |
| KS-AB-03 | Low | Account data are not zeroized during `remove_token` |
| KS-AB-04 | Low | Comment instead of code logic |
| KS-AB-05 | Low | Loss of precision when casting during staking |
| KS-AB-06 | Low | Missing zero address validation |
| KS-AB-07 | Low | NEAR contract address is not forbidden |
| KS-AB-08 | Low | Price caching time is higher than the update time |
| KS-AB-09 | Low | Reentrancy attacks |
| KS-AB-10 | Low | Unlock may be created for unsupported token |
| KS-AB-11 | Low | Vulnerabilities in transitive dependencies |
| KS-AB-12 | Low | Vulnerable dependency |
| KS-AB-13 | Informational | Different roles to start and stop the bridge |
| KS-AB-14 | Informational | Extensive use of `find_program_address` |
| KS-AB-15 | Informational | Inconsistency between comment and code |
| KS-AB-16 | Informational | Missing Ownership Checks |
| KS-AB-17 | Informational | Missing tests of behavior after reward update |
| KS-AB-18 | Informational | Public visibility is set for functions that are not called internally |
| KS-AB-19 | Informational | Vulnerable openzeppelin version |

Table 1: Findings Overview

## Technical Analysis

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

Further investigations concluded that no critical risks were identified for the application, including:

- No potential panics were detected
- No potential errors regarding wraps/unwraps, expect and wildcards
- No internal unintentional unsafe references

### Conclusion

Based on formal verification we conclude that the code implements the documented functionality to the extent of the reviewed code.

# Technical Findings

## General Observations

The Allbridge EVM Contracts project consists of the following contracts: - Bridge - Farming - FeeOracle - Staking - WrappedToken

The overall quality of the code is good. Functions are self-explanatory and comments explain details of the logic. The `Readme.md` file is not finished and lacks description for some parts of the project. Almost all the code is covered with unit tests except for a few functions.

The project was developed with attention to security. During the audit, mostly low-level and informational issues were found. Most of them are safeguards for mistakes. And two medium severity issues were discovered specific to the Bridge Solana Contract. There are several small issues pertaining to the Bridge info server that may affect the security of the system, but in general, the service is safe to use.

Bridge info server is a service that provides information about: - transaction confirmation - supported tokens - supported networks - contract balance - token information - staking pool information

The current version supports Solana, Terra, Near, and EVM networks.

This project's file structure is well done, and the code is divided into separate files in a logical manner. The code style is decent, but there are multiple recommendations from linters that can improve the readability of the code.

Even though the project does not use any frameworks to express account constraints, the developers made sure to perform all critical checks, leaving small room for attacks. The mapping to Ethereum contracts was clear and made auditing the project straightforward.

## Locks and Unlocks can be denied

Finding ID: KS-AB-01
Severity: **Medium**
Status: **Remediated**

### Description

The validator's `CreateLock` and `CreateUnlock` instructions create accounts with PDAs as their final step. Before that, they check that the PDA does not correspond to an account that is already created. The check fails if the account holds lamports. This can be abused by an attacker which can send the minimum amount of lamports required to PDAs and can be computed for specific `lock_id`s.

## New validators can replay previous validators' unlocks

Finding ID: KS-AB-02
Severity: **Medium**
Status: **Accepted at risk by client**

### Description

To make sure `create_unlock` instructions cannot be replayed, the validator program creates an account with a program derived address (PDA) using the `validator_account` key and a `lock_id` as seeds. Furthermore, during signature validation, the signed message does not contain the validator's key. If the address of the bridge's validator is changed via the `SetValidator` instruction, the new validator would re-approve previous unlocks as valid signatures.

## Account data are not zeroized during `remove_token`

Finding ID: KS-AB-03
Severity: **Low**
Status: **Accepted at risk by client**

### Description

When removing tokens, respective `Asset` accounts are deleted. Their deletion involves transferring their remaining lamports. Thus, the solana runtime will delete the account at the end of the transaction. However, instructions within the same transaction will be able to use the `Asset` account with its data.

## Comment instead of code logic

Finding ID: KS-AB-04
Severity: **Low**
Status: **Accepted at risk by client**

### Description

The logic that is crucial for stable workflow should be implemented in the code instead of warning in the comments.

## Loss of precision when casting during staking

Finding ID: KS-AB-05
Severity: **Low**
Status: **Remediated**

### Description

The amount to be transferred while withdrawing or depositing stakes involves a calculation of the form `a*b/c` amongst `u64` types which are elevated to `u128`. At the end of the calculation, the result is cast back to `u64` without any checks, which could potentially cause loss of precision.

## Missing zero address validation

Finding ID: KS-AB-06
Severity: **Low**
Status: **Accepted at risk by client**

### Description

Ensure that a smart contract is initialized correctly to maintain contract ownership. The operation with ETH should check whether the recipient is zero address. All ETH sent to zero address will be lost.

## NEAR contract address is not forbidden

Finding ID: KS-AB-07
Severity: **Low**
Status: **Accepted at risk by client**

### Description

All contract addresses for Solana, Terra, and EVM networks are added to the list of forbidden addresses. The only exception is the contract address of NEAR network.

## Price caching time is higher than the update time

Finding ID: KS-AB-08
Severity: **Low**
Status: **Accepted at risk by client**

### Description

The Bridge info server gets the price from coingecko and caches it for 5 minutes, but on coingecko the price is updated every 1 to 10 minutes.

## Reentrancy attacks

Finding ID: KS-AB-09
Severity: **Low**
Status: **Accepted at risk by client**

### Description

Reentrancy weaknesses occur when an application incorporates untrusted data in a smart contract call without proper validation or the ability to break out. Reentrancy attacks may occur in functions that perform external calls or ETH transferring before storage modification or when an event is emitted. It may lead to funds stealing if an external contract or ETH recipient is untrusted.

## Unlock may be created for unsupported token

Finding ID: KS-AB-10
Severity: **Low**
Status: **Accepted at risk by client**

### Description

The function calls the external contract to “Create message hash and validate the signature” before checking whether the token is supported.

## Vulnerabilities in transitive dependencies

Finding ID: KS-AB-11
Severity: **Low**
Status: **Accepted at risk by client**

### Description

Some transitive dependencies have known vulnerabilities: - follow-redirects <=1.14.7: Exposure of sensitive information - json-schema <0.4.0: Prototype Pollution - node-fetch <2.6.7: Exposure of Sensitive Information to an Unauthorized Actor - node-forge <1.0.0: Prototype Pollution - shelljs <0.8.5: Improper Privilege Management - simple-get <2.8.2: Exposure of Sensitive Information

## Vulnerable dependency

Finding ID: KS-AB-12
Severity: **Low**
Status: **Accepted at risk by client**

### Description

The project uses a vulnerable version of `class-validator`.

### Different roles to start and stop the bridge

Finding ID: KS-AB-13
Severity: **Informational**

#### Description

There are two different roles that can set the state of the bridge

### Extensive use of `find_program_address`

Finding ID: KS-AB-14
Severity: **Informational**

#### Description

According to the Solana documentation for `find_program_address`: > Programs that are meant to be very performant may not want to use this function because it could take a considerable amount of time. There are other means of validating PDAs without the use of `find_program_address`.

### Inconsistency between comment and code

Finding ID: KS-AB-15
Severity: **Informational**

#### Description

The logic described in the comment does not correspond to what the code is doing. The comment says `Mark lock as received` but the code checks if the token is supported.

### Missing Ownership Checks

Finding ID: KS-AB-16
Severity: **Informational**

#### Description

The following accounts are using instructions without properly checking their ownership. The effects of this were investigated.

### Missing tests of behavior after reward update

Finding ID: KS-AB-17
Severity: **Informational**

### Description

The `set` function changes pool's REWARD allocation point that is used in the calculation of rewards and this scenario is not covered with unit tests.

## Public visibility is set for functions that are not called internally

Finding ID: KS-AB-18
Severity: **Informational**

### Description

Public visibility is used for functions that should be accessible from other contracts, via transactions, and from the current contract. Functions that are not meant to be called internally should have External visibility.

## Vulnerable OpenZeppelin version

Finding ID: KS-AB-19
Severity: **Informational**

### Description

The project uses the old version of openzeppelin that has known vulnerabilities.

# METHODOLOGY

Kudelski Security uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

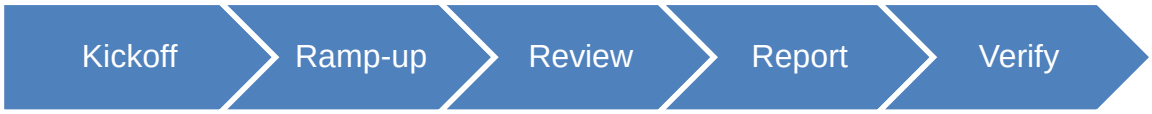


Figure 2: Methodology Flow

## Kickoff

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

## Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the particular project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

## Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

## Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general list and not comprehensive, meant only to give an understanding of the issues we are looking for.

## Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

## Reporting

Kudelski Security delivers a preliminary report in PDF format that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We not only report security issues identified but also informational findings for improvement categorized into several buckets:

- Critical
- High
- Medium
- Low

- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps, that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

## Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes withing a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

## Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessment the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. These is a solid baseline for severity determination.

## The Classification of identified problems and vulnerabilities

There are four severity levels of an identified security vulnerability.

### Critical – vulnerability that will lead to loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

### High - A vulnerability that can lead to loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys

- Badly generated key materials
- Tx signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

### Medium - a vulnerability that hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-revied crypto functions
- Program crashes leaves core dumps or write sensitive data to log files

### Low - Problems that have a security impact but does not directly impact the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

### Informational

- General recommendations

# Tools

The following tools were used during this portion of the test. A link for more information about the tool is provided as well.

Tools used during the code review and assessment

- Rust – cargo tools
- IDE modules for Rust and analysis of source code
- Cargo audit which uses https://rustsec.org/advisories/ to find vulnerabilities cargo.

## RustSec.org

### About RustSec

The RustSec Advisory Database is a repository of security advisories filed against Rust crates published and maintained by the Rust Secure Code Working Group.

### The RustSec Tool-set used in projects and CI/CD pipelines

- `cargo-audit` - audit Cargo.lock files for crates with security vulnerabilities.
- `cargo-deny` - audit Cargo.lock files for crates with security vulnerabilities, limit the usage of particular dependencies, their licenses, sources to download from, detect multiple versions of same packages in the dependency tree and more.