Vesting Contract Security Code Review Assessment

Bonfida

05 May 2021
Version: 1.0

Presented by:
Kudelski Security Research Team
Kudelski Security – Nagravision SA

Corporate Headquarters
Kudelski Security – Nagravision SA
Route de Genève, 22-24
1033 Cheseaux sur Lausanne
Switzerland

DOCUMENT PROPERTIES

Version:	1.0
File Name:	Research_Report_Vesting.docx
Publication Date:	05 May 2021
Confidentiality Level:	For Public Distribution
Document Owner:	Scott Carlson
Document Recipient:	Bonfida Project / Solana Foundation
Document Status:	Approved

TABLE OF CONTENTS

TABLE OF FIGURES

EXECUTIVE SUMMARY

Kudelski Security ("Kudelski"), the cybersecurity division of the Kudelski Group, was engaged by Solana Foundation & Bonfida to conduct an external security assessment in the form of a code review of the Bonfida Vesting contract/program/application.

The assessment was conducted remotely by the Kudelski Security Team. The tests took place between March 15, 2021 to April 30, 2021 and focused on the following objectives:

1. To help the Client to better understand its security posture
2. To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place
3. To identify potential issues and include improvement recommendations

This report summarizes the tests performed and findings in terms of strengths and weaknesses. It also contains detailed descriptions of any discovered vulnerabilities, steps the Kudelski Security Teams took to exploit each vulnerability, and recommendations for remediation.

1.1 Engagement Limitations

The architecture and code review are based on the documentation and code provided by Bonfida. The code resides in a private repository at https://github.com/Bonfida/token-vesting

The reviews are based on the commit hash:

Token-vesting: 7195b1f4fbb5dcb8508d31db9f5a700156ac3459

All third-party libraries were deemed out-of-scope for this review and are expected to work as designed including the SPL. If we found a critical dependency on a Solana component or a third-party library, we did ensure that the current state of the crate included any necessary fixes to known issues.

1.2 Engagement Analysis

This engagement was comprised of a code review including reviewing how the architecture has been implemented as well as any security issues. The architecture implementation review was based on the documentation and the information retrieved through communication between the Bonfida team, Solana Foundation, and the Kudelski Security team. The implementation review concluded that the application implementation is well done, thoroughly designed, and operates as expected/intended.

The code review was conducted by the Kudelski Security team on the code provided by Bonfida, in the form of a Github repository. The code review focused on the handling of secure and private information handling in the code.

As a result of our work, we identified **0 High**, **0 Medium**, **0 Low**, and **4 Informational** findings.

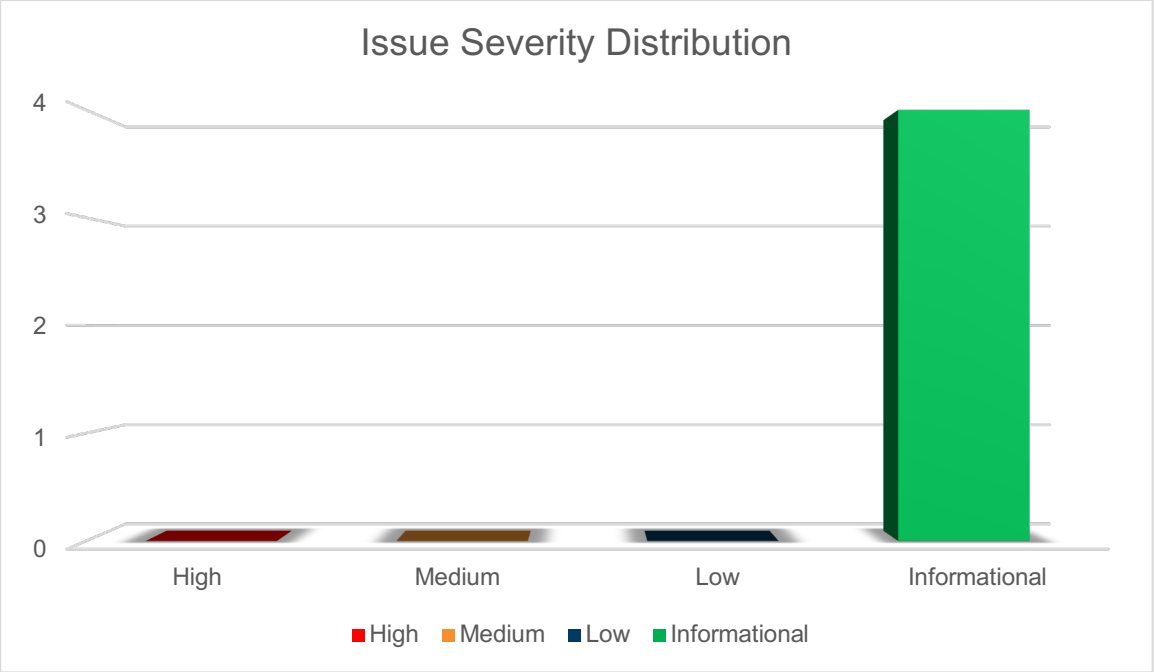

Figure 1 Issue Severity Distribution

1.3 Observations

The code is generally well written and for the most part documented. This facilitates reading of the execution flow. It is worth to mention that there are plenty of hardcoded values that should be re-written as constants. This is only an issue during versioning of the contract if these values need to be changed, versioned, or externalized.

The engagement concluded that the code is fit for the purpose it has been designed for and operates in the bounds of its intended use case.

The use of the Solana SDK in the application is in accordance with the Solana development guidelines, and based on this, we don't see any High or Medium issues in the code provided for the review.

As with any blockchain program, it is important to do error checking to ensure that the data that you pass into the program is what you intend, because programs will gladly attempt to execute transfers between incorrect wallets or with incorrect seeds. It is important that when using this contract, that all documentation is followed and the “caller” ensures that this contract is appropriate for their use case.

1.4 Issue Summary List

ID	SEVERITY	FINDING
KS-Vesting-F-01	**Informational**	Error handling should include clean-up
KS-Vesting-F-02	**Informational**	More error types to be more granular and informative

ID	SEVERITY	FINDING
KS-Vesting-F-03	**Informational**	Use constants to safeguard against unwanted future changes
KS-Vesting-F-04	**Informational**	Use constants to safeguard against unwanted future changes

2. METHODOLOGY

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

Figure 2 Methodology Flow

2.1 Kickoff

The project is started once 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

2.2 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

2.3 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
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 tools, 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

2.4 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:

- 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.

2.5 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.

2.6 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, nor does it cover every possible scenario in which this code can be used. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. These are a solid baseline for severity determination. Information about the severity ratings can be found in **Appendix C** of this document.

3. TECHNICAL DETAILS

This section contains the technical details of our findings as well as recommendations for improvement.

3.1 Error handling should include clean-up

Finding ID: KS-Vesting-F-01

Severity: **Informational**

Status: **Open**

Description

The resulting error is only printed and forwarded to the calling method without any further handling.

Proof of issue

Filename: entrypoint.rs

Beginning Line Number: 10

```
pub fn process_instruction(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    instruction_data: &[u8],
) -> ProgramResult {
    msg!("Entrypoint");
    if let Err(error) = Processor::process_instruction(program_id, accounts, i
nstruction_data) {
        // catch the error so we can print it
        error.print::<VestingError>();
        return Err(error);
    }
    Ok(())
}
```

Severity and Impact Summary

By only printing the error the error is not really handled. If the error has some security implications, it is most likely to go unnoticed.

Recommendation

Make sure to log the error somewhere it can be discovered and processed in case of need.

3.2 More error types to be more granular and informative

Finding ID: KS-Vesting-F-02

Severity: **Informational**

Status: **Open**

Description

There is only one type of error defined in the code base.

Proof of issue

Filename: error.rs

Beginning Line Number: 5

```
/// Errors that may be returned by the Token vesting program.
#[derive(Clone, Debug, Eq, Error, FromPrimitive, PartialEq)]
pub enum VestingError {
    // Invalid instruction
    #[error("Invalid Instruction")]
    InvalidInstruction
}
```

Severity and Impact summary

By only returning one type of error the application may be more difficult to debug.

Recommendation

Look into the possibility to implement more detailed error types upon learning of new failure scenarios.

3.3 Use constants to safeguard against unwanted future changes

Finding ID: KS-Vesting-F-03

Severity: **Informational**

Status: **Open**

Description

Seed arrays are initialized with separate integers. This is present throughout the file.

Proof of issue

Filename: instructions.rs

Beginning Line Number: 71

```
    Init {
        // The seed used to derive the vesting accounts address
        seeds: [u8; 32],
        // The number of release schedules for this contract to hold
        number_of_schedules: u32,
    },
```

Severity and Impact summary

By not using constants it is possible that one of the declarations is unknowingly altered or unknowingly left unaltered. This could lead to incorrect data and/or leaving the application in an unexpected and unwanted state.

Recommendation

Make sure all the seeds declarations are done with a common constant.

3.4 Use constants to safeguard against unwanted future changes

Finding ID: KS-Vesting-F-04

Severity: **Informational**

Status: **Open**

Description

Seed arrays are initialized with separate integers. This is present throughout the file.

Proof of issue

Filename: processors.rs

Beginning Line Number: 28

```
   pub fn process_init(
        program_id: &Pubkey,
        accounts: &[AccountInfo],
        seeds: [u8; 32],
        schedules: u32
    ) -> ProgramResult {
```

Severity and Impact summary

By not using constants it is possible that one of the declarations is unknowingly altered or unknowingly left unaltered. This could lead to incorrect data and/or leaving the application in an unexpected and unwanted state.

Recommendation

Make sure all the seeds declarations are done with a common constant.

APPENDIX A: ABOUT KUDELSKI SECURITY

Kudelski Security is an innovative, independent Swiss provider of tailored cyber and media security solutions to enterprises and public sector institutions. Our team of security experts delivers end-to-end consulting, technology, managed services, and threat intelligence to help organizations build and run successful security programs. Our global reach and cyber solutions focus is reinforced by key international partnerships.

Kudelski Security is a division of Kudelski Group. For more information, please visit https://www.kudelskisecurity.com.

Kudelski Security

route de Genève, 22-24

1033 Cheseaux-sur-Lausanne

Switzerland

Kudelski Security

5090 North 40th Street

Suite 450

Phoenix, Arizona 85018

APPENDIX B: DOCUMENT HISTORY

VERSION	STATUS	DATE	AUTHOR	COMMENTS
0.1	Draft	3 May 2021	Peter Löfgren	Draft to QA
1.0	Final Release	5 May 2021	Scott Carlson	Final Draft
		Select the Date		

REVIEWER	POSITION	DATE	VERSION	COMMENTS
Mikael Björn	Tech Lead	4 May 2021	0.1	Draft
Scott Carlson	Head of Blockchain Security	4 May 2021	0.2	Draft
		Select the Date		

APPROVER	POSITION	DATE	VERSION	COMMENTS
Bonfida Project		4 May 2021	1.0	
		Select the Date		
		Select the Date		

APPENDIX C: SEVERITY RATING DEFINITIONS

Kudelski Security uses a custom approach when determining criticality of identified issues. This is meant to be simple and fast, providing customers with a quick at a glance view of the risk an issue poses to the system. As with anything risk related, these findings are situational. We consider multiple factors when assigning a severity level to an identified vulnerability. A few of these include:

- Impact of exploitation
- Ease of exploitation
- Likelihood of attack
- Exposure of attack surface
- Number of instances of identified vulnerability
- Availability of tools and exploits

SEVERITY	DEFINITION
High	The identified issue may be directly exploitable causing an immediate negative impact on the users, data, and availability of the system for multiple users.
Medium	The identified issue is not directly exploitable but combined with other vulnerabilities may allow for exploitation of the system or exploitation may affect singular users. These findings may also increase in severity in the future as techniques evolve.
Low	The identified issue is not directly exploitable but raises the attack surface of the system. This may be through leaking information that an attacker can use to increase the accuracy of their attacks.
Informational	Informational findings are best practice steps that can be used to harden the application and improve processes.