Chainlink Fluxaggregator code review

Octopus Network

07 July 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_Chainlink-Fluxagg-Final.docx
Publication Date:	07 July 2021
Confidentiality Level:	For Public Disclosure
Document Owner:	Scott Carlson
Document Recipient:	Chainlink project team
Document Status:	Approved

TABLE OF CONTENTS

TABLE OF FIGURES

EXECUTIVE SUMMARY

Kudelski Security ("Kudelski"), the cybersecurity division of the Kudelski Group, was engaged by Octopus Network and the Solana Foundation to conduct an external security assessment in the form of a code review of the Chainlink Fluxaggregator application.

The assessment was conducted remotely by the Kudelski Security Team from our secure lab environment. The tests took place between Mars 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 the 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 Octopus Network. The code resides in a private repository at https://github.com/octopus-network/solana-flux-aggregator

The reviews are based on the commit hash:

solana-flux-aggregator: a224f8ba27e6c96ad6f78227278ed81a583af787

All third-party libraries were deemed out-of-scope for this review and are expected to work as designed. We have when needed based on the criticality of the dependency looked at the current state of the crate included.

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 Octopus Network team and the Kudelski Security team. The implementation review concluded that the application implementation is as good as expected.

The code review was conducted by the Kudelski Security team on the code provided by Octopus Network, 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**, **1 Low**, and **7 Informational** findings.

The only issues found in the code were Low/Informational findings. This shows that the functional level of the application is good and that the risk profile of the application is relatively low.

The findings referred to in the Findings section are such as they would improve the functionality and performance of the application and secure it further.

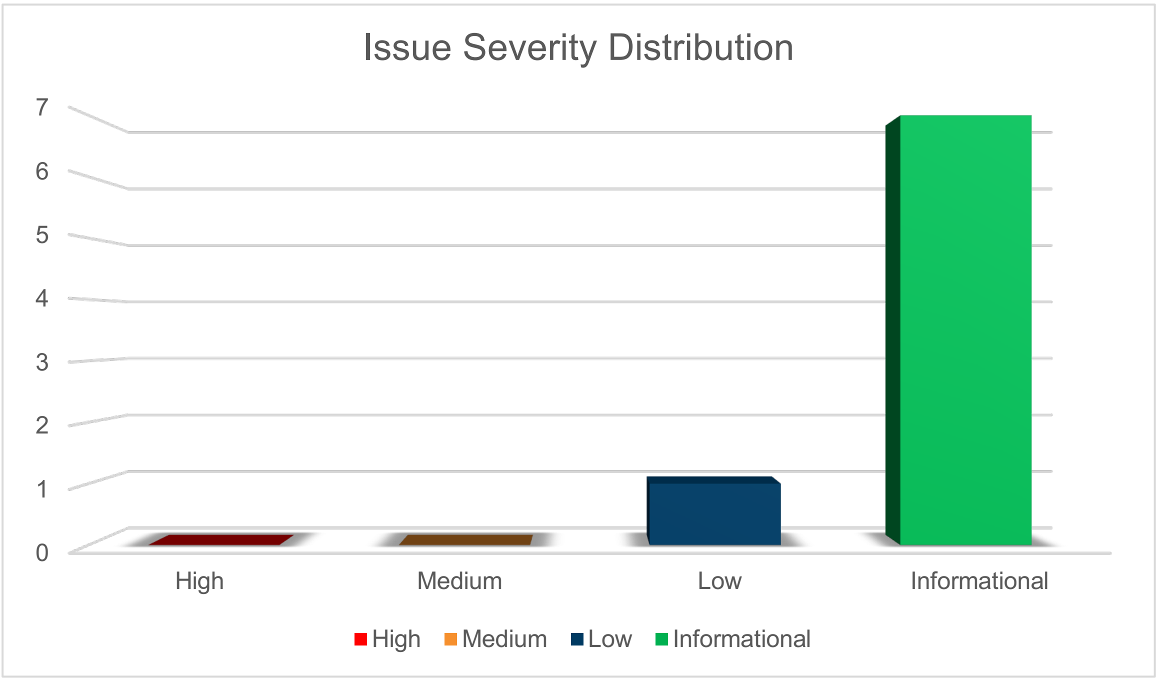


Figure 1 Issue Severity Distribution

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.

The engagement concluded that the code is fit for the purpose it has been designed for.

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 issues in the code provided for the review.

1.4 Issue Summary List

ID	SEVERITY	FINDING
KS-Chainlink-F-01	**Low**	Should use constant to define size
KS-Chainlink-F-02	**Informational**	Unresolved FIXME left in the code
KS-Chainlink-F-03	**Informational**	Unresolved FIXME left in the code
KS-Chainlink-F-04	**Informational**	Function could use constants as return values
KS-Chainlink-F-05	**Informational**	Not all errors are commented

ID	SEVERITY	FINDING
KS-Chainlink-F-06	**Informational**	Could use constant in definition
KS-Chainlink-F-07	**Informational**	Authorization is sufficient
KS-Chainlink-F-08	**Informational**	Authorization for withdraw is sufficient

2. METHODOLOGY

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

Kickoff
Ramp-up
Review
Report
Verify

Figure 2 Methodology Flow

2.1 Kickoff

The project is kicked all of 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 a majority 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 as a whole. 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. 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. 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 Should use constant to define size

Finding ID: KS-Chainlink-F-01

Severity **Low**

Status: **Open**

Description

When defining the structs AddRequesterContext and AddOracleContext int processor.rs and Instruction.AddOracle and Instruction.AddRequester size is defined with separate integers.

Filename: instruction.rs

Beginning Line number: 20

```
pub enum Instruction {
    Initialize {
        config: AggregatorConfig,
    },

    Configure {
        config: AggregatorConfig,
    },

    AddOracle {
        description: [u8; 32],
    },

    RemoveOracle,

    AddRequester {
        description: [u8; 32],
    },

    RemoveRequester,

    RequestRound,

    Submit {
        round_id: u64,
        value: u64,
    },

    Withdraw {
        faucet_owner_seed: Vec<u8>,
    },
}
```

Filename: processor.rs

Beginning line number: 103, 156

```
struct AddOracleContext<'a> {
    rent: Rent,
    aggregator: &'a AccountInfo<'a>,
    aggregator_owner: &'a AccountInfo<'a>, // signed
    oracle: &'a AccountInfo<'a>,
    oracle_owner: &'a AccountInfo<'a>,

    description: [u8; 32],
}
struct AddRequesterContext<'a> {
    rent: Rent,
    aggregator: &'a AccountInfo<'a>,
    aggregator_owner: &'a AccountInfo<'a>, // signed
    requester: &'a AccountInfo<'a>,
    requester_owner: &'a AccountInfo<'a>,

    description: [u8; 32],
}
```

Severity and Impact summary

By not using a constant for defining the size of description there is a possibility for one of the value either being unintentionally altered or one of the values forgotten to be altered. This would probably result in an undesirable application state.

Recommendation

Alter the code to use a constant when defining the size of the description arrays in OracleContext and RequstContext

3.2 Unresolved FIXME left in the code

Finding ID: KS-Chainlink-F-02

Severity: **Informational**

Status: **Open**

Description

There is a block of comment stating that something needs to be fixed, but the code works as intended.

Filename: borsh_state.rs

Beginning line number: 14

```
fn save(&self, account: &AccountInfo) -> ProgramResult {
        let data = self
            .try_to_vec()
            .map_err(|_| ProgramError::InvalidAccountData)?;

        // FIXME: looks like there is association precedence issue that prevents
        // RefMut from being automatically dereferenced.
        //
        // let dst = &mut account.data.borrow_mut();
        //
        // Why does it work in an SPL token program though?
        //
        // Account::pack(source_account, &mut source_account_info.data.borrow_mut())?;
        let mut dst = (*account.data).borrow_mut();
        if dst.len() != data.len() {
            return Err(ProgramError::InvalidAccountData);
        }
        dst.copy_from_slice(&data);

        Ok(())
    }
```

Severity and Impact Summary

If the unresolved FIXME is only documented in code there is a risk that it is forgotten and left unresolved. The code works as intended but the FIXME confuses the reader.

Recommendation

Make sure that the FIXME is documented in the project management tool instead of in the code.

3.3 Unresolved FIXME left in the code

Finding ID: KS-Chainlink-F-03

Severity: **Informational**

Status: **Open**

Description

There is a block of comment stating that something needs to be fixed, but the code works as intended.

Filename: borsh_state.rs

Beginning line number: 36

```
fn save_exempt(&self, account: &AccountInfo, rent: &Rent) -> ProgramResult {
        let data = self
            .try_to_vec()
            .map_err(|_| ProgramError::InvalidAccountData)?;

        if !rent.is_exempt(account.lamports(), data.len()) {
            // FIXME: return a custom error
            return Err(ProgramError::InvalidAccountData);
        }

        let mut dst = (*account.data).borrow_mut();
        if dst.len() != data.len() {
            // FIXME: return a custom error
            return Err(ProgramError::InvalidAccountData);
        }
        dst.copy_from_slice(&data);

        Ok(())
    }
```

Severity and Impact Summary

If the unresolved FIXME is only documented in code there is a risk that it is forgotten and left unresolved. The code works as intended but the FIXME confuses the reader.

Recommendation

Make sure that the FIXME is documented in the project management tool instead of in the code.

3.4 Function could use constants as return values

Finding ID: KS-Chainlink-F-04

Severity: **Informational**

Status: **Open**

Description

Function does not use constants as return values.

Filename: borsh_utils.rs

Beginning line number: 6

```rust
/// Get packed length for the given BorchSchema Declaration
fn get_declaration_packed_len(
    declaration: &str,
    definitions: &HashMap<Declaration, Definition>,
) -> usize {
    match definitions.get(declaration) {
        Some(Definition::Array { length, elements }) => {
            *length as usize * get_declaration_packed_len(elements, definitions)
        }
        Some(Definition::Enum { variants }) => {
            1 + variants
                .iter()
                .map(|(_, declaration)| get_declaration_packed_len(declaration, definitions))
                .max()
                .unwrap_or(0)
        }
        Some(Definition::Struct { fields }) => match fields {
            Fields::NamedFields(named_fields) => named_fields
                .iter()
                .map(|(_, declaration)| get_declaration_packed_len(declaration, definitions))
                .sum(),
            Fields::UnnamedFields(declarations) => declarations
                .iter()
                .map(|declaration| get_declaration_packed_len(declaration, definitions))
                .sum(),
            Fields::Empty => 0,
        },
        Some(Definition::Sequence {
            elements: _elements,
        }) => panic!("Missing support for Definition::Sequence"),
        Some(Definition::Tuple { elements }) => elements
            .iter()
            .map(|element| get_declaration_packed_len(element, definitions))
            .sum(),
        None => match declaration {
            "u8" | "i8" => 1,
            "u16" | "i16" => 2,
            "u32" | "i32" => 4,
            "u64" | "i64" => 8,
            "u128" | "i128" => 16,
            "bool" => 1,
            "nil" => 0,
            _ => panic!("Missing primitive type: {}", declaration),
        },
    }
}
```

Severity and Impact Summary

By not using constants there is a higher risk that the returned value is unintentionally altered resulting in an erroneous being return resulting in an unexpected application state.

Recommendation

Define constants for the return values to prevent erroneous values being returned.

3.5 Not all errors are commented

Finding ID: KS-Chainlink-F-05

Severity: **Informational**

Status: **Open**

Description

Not all errors are commented with an integer

Filename: error.rs

Beginning line number: 43

```
    #[error("No resolve answer")]
    NoResolvedAnswer,

    #[error("No submitted value")]
    NoSubmission,

    #[error("Invalid faucet")]
    InvalidFaucet,

    #[error("Unknown error")]
    UnknownError,
```

Severity and Impact Summary

By not commenting all the errors the application may harder to debug due to the fact it may be harder to identify which error is being thrown.

Recommendation

Add comments to the remaining four errors.

3.6 Could use constant in definition

Finding ID: KS-Chainlink-F-06

Severity: **Informational**

Status: **Open**

Description

Lamports are defined with integers directly.

Filename: processor.rs

Beginning line number: 601

```
fn rent_sysvar() -> TSysAccount {
        TSysAccount(sysvar::rent::id(), create_account(&Rent::default(), 42))
    }

fn sysclock(time: i64) -> TSysAccount {
        let mut clock = Clock::default();
        clock.slot = time as u64;
        TSysAccount(sysvar::clock::id(), create_account(&clock, 42))
    }
```

Severity and Impact Summary

By not using a constant the test may be unintentionally altered to use incorrect values resulting in incorrect test results.

Recommendation

Change the code to use constant to make sure the same number of lamports always are used

3.7 Authorization is sufficient (Informational)

The general authorization mechanism used in Re Chainlink Flux is:

1. Given an account to do something with
2. The accounts's owner/authority must have signed the transaction.

This authorization mechanism is used for processing the following instructions.

Instructions authorized by aggregator accounts:

- Initialize -- register aggregator owner
- Configure -- register aggregator SPL token account used in withdraw
- Add oracle -- register oracle owner
- Remove oracle
- Add requester -- register requester owner
- Remove requester

Instructions authorized by oracle accounts:

- Submit
- Withdraw

Instructions authorized by requester accounts:

- Request round

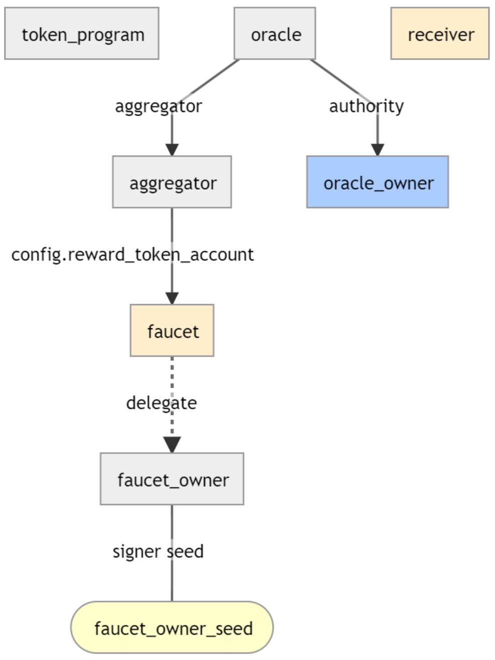
Furthermore, the references from the account "to do something with" are checked against the accounts provided as input.

For withdraw the signing seed for the SPL token account to transfer tokens from is required. Thus, the program has no signing authority to withdraw that can be abused.

Thus, the authorization mechanism looks sound.

3.8 Authorization for withdraw is sufficient (Informational)

WITHDRAW



The authorization for the withdraw instruction requires the owner of the `oracle` to sign the transaction. Furthermore, the signer seed for the delegate authority of the `facet` (source) account must also be precent.

As the delegate authority's signer seeds are required it is not possible to do unauthorized withdraws.

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	Fredrik Strander	Draft to QA
0.3	Final Draft	11 June 2021	Scott Carlson	
1.0	Final For Public Release	7 July 2021	Scott Carlson	Final

REVIEWER	POSITION	DATE	VERSION	COMMENTS
Mikael Björn	Tech Lead	4 May 2021	0.1	Draft
		Select the Date		
		Select the Date		

APPROVER	POSITION	DATE	VERSION	COMMENTS
		Select the Date		
		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.