FYEO

Security Assessment of buddlink-beta-sol

Buddy Link

August 2023
Version 1.0

Presented by:
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Security Level
Public

Table of Contents

Executive Summary 2
- Overview 2
- Key Findings 2
- Scope and Rules of Engagement 3

Technical Analyses and Findings 4
- Findings 5
- Technical Analysis 5
- Conclusion 5

Technical Findings 6
- General Observations 6
- Consider using a configurable value for the `SUPER_ADMIN_PUBKEY`, etc. global constants 7
- `is_valid_profile_name_string`, `set_padding` and related functions have incorrect assumptions 8
- Accounts lack any versioning or path to upgradability 9
- Document contract methods 10
- Duplicate code in claim logic 11
- Use `account.exit()` instead of manual shim 12
- `match_owners_pubkey` and `match_owners` could be rewritten more efficiently 13

Our Process 15
- Methodology 15
  - Kickoff 15
  - Ramp-up 15
  - Review 16
  - Code Safety 16
  - Technical Specification Matching 16
  - Reporting 17
  - Verify 17
- Additional Note 17
- The Classification of vulnerabilities 18

List of Figures

Figure 1: Findings by Severity 4

Figure 2: Methodology Flow 15

List of Tables

Table 1: Scope3

Table 2: Findings Overview5

Table 3: Legend for relationship graphs 11

EXECUTIVE SUMMARY

Overview

Buddy Link engaged FYEO Inc. to perform a Security Assessment of buddlink-beta-sol.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on August 08 - August 23, 2023, and focused on the following objectives:

- To 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 results of our tests.

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

Key Findings

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

- FYEO-TFLM-ID-01 – Consider using a configurable value for the `SUPER_ADMIN_PUBKEY`, etc. global constants
- FYEO-TFLM-ID-02 – `is_valid_profile_name_string`, `set_padding` and related functions have incorrect assumptions
- FYEO-TFLM-ID-03 – Accounts lack any versioning or path to upgradability
- FYEO-TFLM-ID-04 – Document contract methods
- FYEO-TFLM-ID-05 – Duplicate code in claim logic
- FYEO-TFLM-ID-06 – Use `account.exit()` instead of manual shim
- FYEO-TFLM-ID-07 – `match_owners_pubkey` and `match_owners` could be rewritten more efficiently

Based on our review process, we conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

The FYEO Review Team performed a Security Assessment of buddlink-beta-sol. 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/ladder-caster/buddy-link-beta-sol with the commit hash db6aeb650b45d4875c0d7b5272f820182e5a0d81.

Files included in the code review
`buddy-link-beta-sol/`

Table 1: Scope

TECHNICAL ANALYSES AND FINDINGS

During the Security Assessment of buddlink-beta-sol, we discovered:

- 1 finding with MEDIUM severity rating.
- 1 finding with LOW severity rating.
- 5 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

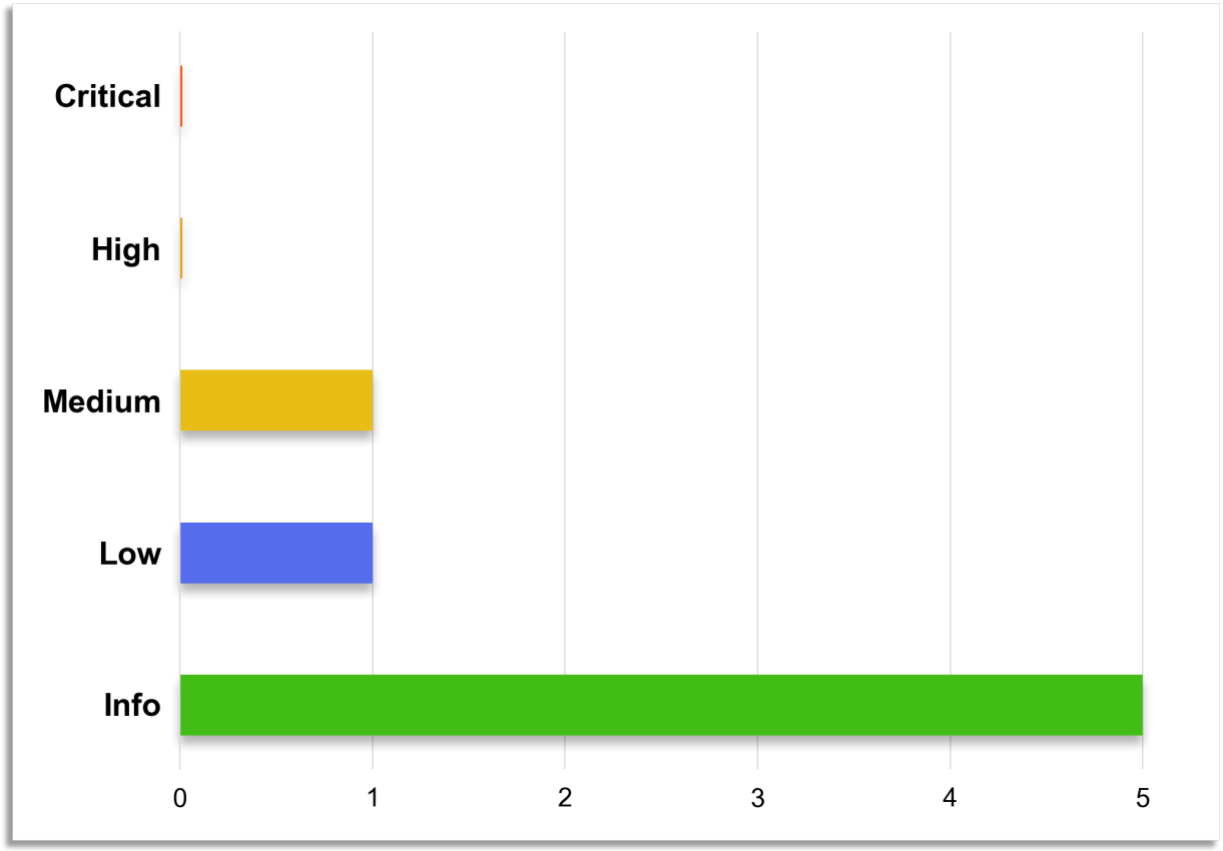


Figure 1: Findings by Severity

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.

Finding #	Severity	Description
FYEO-TFLM-ID-01	**Medium**	Consider using a configurable value for the `SUPER_ADMIN_PUBKEY`, etc. global constants
FYEO-TFLM-ID-02	**Low**	`is_valid_profile_name_string`, `set_padding` and related functions have incorrect assumptions
FYEO-TFLM-ID-03	**Informational**	Accounts lack any versioning or path to upgradability
FYEO-TFLM-ID-04	**Informational**	Document contract methods
FYEO-TFLM-ID-05	**Informational**	Duplicate code in claim logic
FYEO-TFLM-ID-06	**Informational**	Use `account.exit()` instead of manual shim
FYEO-TFLM-ID-07	**Informational**	`match_owners_pubkey` and `match_owners` could be rewritten more efficiently

Table 2: 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.

CONCLUSION

Based on our review process, we conclude that the code implements the documented functionality to the extent of the reviewed code.

Technical Findings

General Observations

The code is well-structured and easy to read and parse. It is clear that it was written with an understanding of both Rust-specific patterns and the best practices unique to the Solana ecosystem. While several small issues were found, they do not considerably detract from the otherwise fine code.

Architecturally, the contract appears sound and all checks one might expect are performed where necessary.

CONSIDER USING A CONFIGURABLE VALUE FOR THE `SUPER_ADMIN_PUBKEY`, ETC. GLOBAL CONSTANTS

Finding ID: FYEO-TFLM-ID-01
Severity: **Medium**
Status: **Remediated**

Description

The current contract implementation contains a variety of hard-coded global values.

Proof of Issue

File name: `programs/buddylink/src/utils/constants/shared.rs`

```
pub const SUPER_ADMIN_PUBKEY: Pubkey =
pubkey!("Cndn5vqZrPRQu5BSZ9ovSjnLYLekn2pRv8uwZdKNgKqM");
```

Severity and Impact Summary

In the case that the global administrator key is for any reason compromised, the current implementation does not permit a replacement key to be installed.

Recommendation

A mechanism to replace the administrator key, and possibly also freeze all operations on a contract in case of an emergency, would prevent a situation where a known loss of key security leads to a compromise of the whole contract.

The permissions to change the administrator key could be gated behind a multisig or similar arrangement.

`IS_VALID_PROFILE_NAME_STRING`, `SET_PADDING` AND RELATED FUNCTIONS HAVE INCORRECT ASSUMPTIONS

Finding ID: FYEO-TFLM-ID-02
Severity: **Low**
Status: **Remediated**

Description

Functions related to validating string input for names do not correctly handle large UTF-8 characters.

`set_padding` assumes that name strings are no longer than 36 bytes. However, `name.chars().count()` only counts UTF-8 *codepoints*, and each codepoint can be up to 4 bytes long. With `PROFILE_NAME_LENGTH = 18`, this means that the max size of a name is actually 4 * 18 = 72 bytes.

Proof of Issue

File name: `programs/buddylink/src/utils/string_util.rs`
Line number: 22

```
pub fn is_valid_profile_name_string(name: &str) -> bool {
    let char_count = name.chars().count();

    name.chars().all(|c| c.is_alphanumeric()) && char_count == PROFILE_NAME_LENGTH
}
```

File name: `programs/buddylink/src/utils/accounts/buddy_util.rs`
Line number: 18

```
    pub fn set_padding(&mut self) {
        self.padding = vec![0; 40 - (4 + self.name.len())];
    }
```

Severity and Impact Summary

This might lead to some hard to debug panics, but otherwise does not constitute a security issue.

Recommendation

Assume that each codepoint can be up to 4 bytes long and handle strings with that assumption.

ACCOUNTS LACK ANY VERSIONING OR PATH TO UPGRADABILITY

Finding ID: FYEO-TFLM-ID-03
Severity: **Informational**
Status: **Remediated**

Description

Current architecture does not presume any potential upgrades or changes to the contract accounts.

Proof of Issue

File name: N/A
Line number: N/A

Severity and Impact Summary

This could be a potential footgun for the future maintainer wishing to extend the contract with extra functionality without breaking backwards compatibility.

Recommendation

Consider adding at least a `version` field to account structures. It is fine if the current implementation simply ignores the version, but this opens the door to enable versioning down the line if desirable.

DOCUMENT CONTRACT METHODS

Finding ID: FYEO-TFLM-ID-04
Severity: **Informational**
Status: **Remediated**

Description

Contract methods are mostly undocumented in their intent, aside from the identifier names.

Proof of Issue

File name: `programs/buddylink/src/lib.rs`
Line number: N/A

Severity and Impact Summary

Informational

Recommendation

It would help any future maintainers and/or reviewers to have the following context for all relevant methods: * What is the primary purpose of the method? * Who is the intended caller of the method? * What important access control is supposed to be performed in the method? * What non-obvious side-effects may the method introduce?

DUPLICATE CODE IN CLAIM LOGIC

Finding ID: FYEO-TFLM-ID-05
Severity: **Informational**
Status: **Remediated**

Description

Arithmetic code related to claiming logic is duplicated across multiple functions.

Proof of Issue

File name:
programs/buddylink/src/instructions/claim/no_multi_level/claim.rs
programs/buddylink/src/instructions/claim/no_multi_level/transfer_up.rs
programs/buddylink/src/instructions/claim/claim.rs
programs/buddylink/src/instructions/claim/transfer_up.rs

Line number: N/A

```
    let (claiming_buddy_share, claimed_by_buddy) =
        buddy_treasury.get_treasury_owned_by_buddy_info(&buddy.key(), true);

    let claiming_buddy_amount: u64 = available_amount
        .checked_sub(buddy_treasury.get_total_frozen(true))
        .unwrap()
        .checked_add(buddy_treasury.get_total_claimed(true))
        .unwrap()
        .checked_mul(claiming_buddy_share)
        .unwrap()
        .checked_div(buddy_treasury.get_total_shares())
        .unwrap()
        .checked_sub(claimed_by_buddy)
        .unwrap();

    // etc...
```

Severity and Impact Summary

Informational

Recommendation

Factoring out this logic would ensure that future changes touching the claim logic do not accidentally miss any of the sites where this logic is referenced.

USE `ACCOUNT.EXIT()` INSTEAD OF MANUAL SHIM

Finding ID: FYEO-TFLM-ID-06
Severity: **Informational**
Status: **Remediated**

Description

`write_data_manually` is a reimplementation of `account.exit()`

Proof of Issue

File name:
`programs/buddylink/src/instructions/rewards/transfer_reward_unchecked_multiple.rs`
Line number: 124

```
write_data_manually(&mut member.to_account_info(), member);
```

Severity and Impact Summary

Informational

Recommendation

Use `member.exit()` as the canonical way to perform a persistence flush.

`MATCH_OWNERS_PUBKEY` AND `MATCH_OWNERS` COULD BE REWRITTEN MORE EFFICIENTLY

Finding ID: FYEO-TFLM-ID-07
Severity: **Informational**
Status: **Remediated**

Description

`match_owners_pubkey` and `match_owners` make multiple passes over the same data and could be improved to save some gas.

Proof of Issue

File name: `programs/buddylink/src/utils/accounts/treasury_util.rs`
Line number: 34-54

```
 pub fn match_owners(&self, owners_to_match: &[TreasuryOwner]) -> bool {
        self.owners.iter().all(|owner1| {
            owners_to_match
                .iter()
                .any(|owner2| owner1.owner_pda == owner2.owner_pda)
        }) && owners_to_match.iter().all(|owner2| {
            self.owners
                .iter()
                .any(|owner1| owner2.owner_pda == owner1.owner_pda)
        })
    }

    pub fn match_owners_pubkey(&self, owners_to_match: &[Pubkey]) -> bool {
        self.owners.iter().all(|owner1| {
            owners_to_match
                .iter()
                .any(|owner2| owner1.owner_pda == *owner2)
        }) && owners_to_match
            .iter()
            .all(|owner2| self.owners.iter().any(|owner1| *owner2 ==
owner1.owner_pda))
    }
```

Severity and Impact Summary

Informational

Recommendation

We can abuse the fact that `self.owners` does not contain duplicates, and only do a single pass over the data like this:

```
        let count = self.owners
            .iter()
```

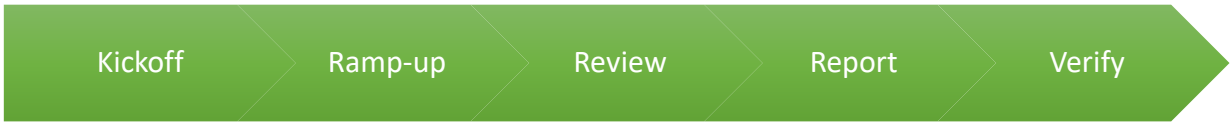
```
            .filter(|s| owners_to_match.contains(&s.owner_pda))
            .count();
        count == owners_to_match.len() && count == self.owners.len()
```

OUR PROCESS

METHODOLOGY

FYEO Inc. 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 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 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

FYEO Inc. delivers a draft report 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 report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- 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 within 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 assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.

THE CLASSIFICATION OF VULNERABILITIES

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a 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 - vulnerability has potential to lead to a 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
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability 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-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

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

Informational

- General recommendations