Smart Contract Secure Code Review

Findings and Recommendations Report Presented to:

Beluga

September 07, 2022
Version: 1.1

Presented by:

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

PUBLIC RELEASE

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

Overview

Beluga engaged Kudelski Security to perform a Smart Contract Secure Code Review.

The assessment was conducted remotely by the Kudelski Security Team. Testing took place from July 25 to August 22, 2022 with a retest completed on September 7, 2022. These tests focused on the following objectives:

- Provide the customer with an assessment of the Beluga DEX and Beluga DEX-Core and its 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 are the major themes and issues identified during the testing period. These, along with other items, within the findings section, were prioritized for remediation and remediated accordingly.

- Bump Seed Canonicalization – The program does not validate the `bump_seed` parameter that is supplied through the `authority_id` function call.
- Missing Empty String Check – The program does not check for valid string values when importing environment variables
- Missing tests – Newly developed code and some features did not include adequate tests to validate both the functionality and security of the application.

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

- Selection of *solana-program-library* as the starting point of the application inherits a number of security protections.
- The reviewed code bases responded well to traditional injection and other OWASP-style attacks, with a few exceptions.

Scope and Rules of Engagement

Kudelski performed a Smart Contract Secure Code Review for Beluga. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

In-Scope Code Repository	
https://github.com/Belugadex/Belugadex-core	
Code Commit	
Initial test commit - 28a7ac585e394fee7d75af93d84816eecd66ff0d Re-test commit - 8de0d50630cef2079559de7a524df768d301cc8f	
In-Scope Applications	
Application	Purpose
Beluga DEX – Beluga DEX-Core	A Decentralized Exchange for Stable Coins and Pegged Assets

Table 1: Scope

TECHNICAL ANALYSIS & FINDINGS

During the Smart Contract Secure Code Review, we discovered three (3) medium-severity findings, as well as one (1) low-severity finding. Retesting of the identified findings discovered that all issues rated low or medium severity were resolved.

The following chart displays the findings by severity during the initial analysis.

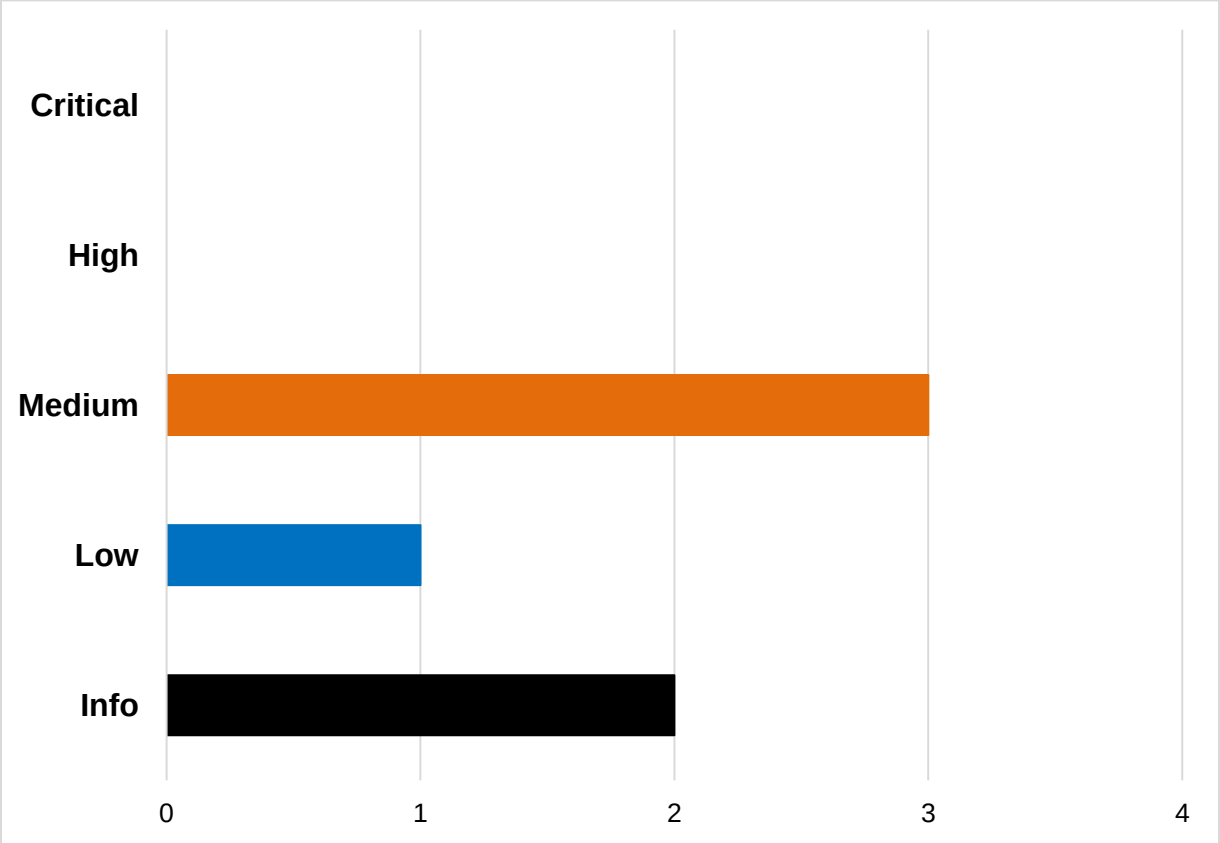

Figure 1: Findings by Severity

Threat Analysis

This threat analysis section summarizes the threat scope and key threats identified during the code review, which informed the secure code review analysis. It also contains descriptions of the threats discovered and potential vulnerabilities as well as any applicable recommendations for remediation.

Threat Scope

Kudelski utilized the following Solana program application architecture descriptions and the provided source code to identify threat boundaries, threat actors, and quantify possible threats to the provided application source, its infrastructure, and supporting processes. To further refine this activity, threat analysis to the various application components was scoped to actors targeting the provided codebase.

Various trust boundaries were identified in the source code and Beluga and Solana developer documentation, with special attention focused on internet-accessible boundaries, including user to program interactions.

Threat Actors

During the secure code review, Kudelski considered several different threat actors that could target the application. Of the identified threat actors, malicious external attackers were considered most likely to target various application processes, users, and personnel. This is followed by malicious insiders such as developers and anonymous internal and external users.

Threat Actor	Observed Risk
Anonymous External Attacker	Low
Anonymous Internal Attacker	Low
Malicious External User	High
Malicious Internal User	Low
Malicious Employee	Medium
Malicious Administrator	Low
Malicious Developer	High
Nation State Actor	Low

Key Threats

- Manipulation of user-controlled addresses and amounts to bypass expected program controls.
- Use of outdated libraries and dependencies could introduce unexpected vulnerabilities and risk to the program, especially given deployment to the public blockchain.
- As discussed in Beluga documentation, there is an impermanent-loss risk related to point-in-time capital reflecting a lower value when tokens are committed to liquidity pools relative to the value of tokens simply held in reserve.
- Publicly available functions within the program could be called unexpectedly or out-of-order to cause confusion or bypass proper program checks.

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
1	~~**Medium**~~ **Resolved**	Bump Seed Canonicalization
2	~~**Medium**~~ **Resolved**	Missing Empty String Check
3	~~**Medium**~~ **Resolved**	Missing Tests
4	~~**Low**~~ **Resolved**	Using Components with Known Vulnerabilities
5	**Informational**	Using Rust Nightly Compiler
6	~~**Informational**~~ **Resolved**	Incomplete Code

Table 2: Findings Overview

1 – Bump Seed Canonicalization

Severity	Resolved

Impact	Likelihood	Difficulty
Medium	Medium	Moderate

Description

The program does not validate the `bump_seed` parameter that is supplied through the `authority_id` function call.

2 – Missing Empty String Check

Severity	Resolved

Impact	Likelihood	Difficulty
Medium	Low	Moderate

Description

The program does not check for valid string values when importing sensitive environment variables, including the address where program gas fees are sent.

3 – Missing Tests

Severity	Resolved

Impact	Likelihood	Difficulty
Medium	High	Moderate

Description

The reviewed source code contained some unit, integration, and fuzzing tests based upon the *solana-program-library* implementation, but newly developed code and some features did not include adequate tests to validate both the functionality and security of the application. In addition, the reference implementation provided in the *solana-program-library* included more test helpers and tests for curves than was included in the reviewed source.

4 – Using Components with Known Vulnerabilities

Severity	Resolved

Impact	Likelihood	Difficulty
Low	High	Moderate

Description

Outdated or weak components are in use by the application. These components may be part of a programming library or underlying platform. These weaknesses are commonly targeted by attackers because of the publicly available information on these vulnerabilities.

5 – Using Rust Nightly Compiler

Severity	INFORMATIONAL

Impact	Likelihood	Difficulty
Informational	Low	High

Description

The smart contract build process allowed for use of the `rust-nightly` binary to build the contract for deployment.

6 – Incomplete Code

Severity	Resolved

Description

The program has stubbed out code that does not have any functionality beyond returning `Ok(())`. The program also contains `// TODO` comments relating to packing/unpacking and validation functionality.

METHODOLOGY

Approach

Kudelski utilizes a standard methodology for assessments that is comprised of three phases: information gathering, vulnerability identification, and reporting. Each phase feeds the next, but any activity in later phases may inform additional research and testing. The activities are cyclical to provide the analyst with working knowledge of the targeted properties for additional threat vectors.

Security methods in Cryptocurrency and Cryptocurrency Exchanges

In analyses of the threat vectors facing Cryptocurrency applications, source code, and exchanges, Kudelski Security uses a testing regimen that follows a best-practices heuristic recognizing five likely areas of security weaknesses specific to cryptocurrency: 1) Susceptibility to phishing; 2) Weak hot wallet protections; 3) Broken Authorization class vulnerabilities related to login credentials of individuals with privileged roles; 4) Software vulnerabilities; and 5) Transaction malleabilities.

General security checklists for Cryptocurrency Blockchain vulnerabilities are further informed by chain-specific security concerns. In security reviews of Solana platform applications and codebases, Kudelski reviews security controls relative to the eleven vulnerabilities laid out in coral-xyz's Sealevel Attacks github project: (1) Signer Authorization; (2) Account Data Matching, (3) Owner Checks, (4) Type Cosplay, (5) Initialization, (6) Arbitrary CPI, (7) Duplicate Mutable Accounts, (8) Bump-Seed Canonicalization, (9) PDA Sharing, (10) Closing Accounts, (11) sysvar Address Matching.

Kudelski continually updates reviewers' knowledge base relative to blockchain vulnerabilities in languages under review in open acknowledgment of Web3's rapidly evolving context.

Information Gathering

Kudelski starts by reviewing application endpoints based on availability, application use-cases, developer documentation, and application source code. These endpoints are analyzed for use, potential parameters, additional attack surface, and possible threats. Applications are reviewed during this phase from multiple points of view, including an anonymous, un-authenticated user, an authenticated user, and an authenticated partner.

Kudelski analyzes available endpoints and source code during this phase for controls that affect security posture, including authentication and authorization controls, logging behavior, communication protocols, input handling, encryption settings, and other application behavior.

Vulnerability Identification

Kudelski uses the identified endpoints and controls of the identified assets to identify and explore possible security vulnerabilities across applications based on our expertise in assessing application flaws. Special attention will be paid to possible fraud and business logic flaws that could affect the Client, its partners, or its customers.

Kudelski utilizes industry-standard vulnerability lists for assessment purposes, including OWASP's Application Security Verification Standard, the OWASP Top 10 Security Risks, and the SANS CWE Top 25 Software Errors. These vulnerabilities are assessed across various security domains as they apply to the targeted application. Additional attack surfaces and weaknesses may be noted during this portion of the assessment for further research.

Security methods for assessing Decentralized Cryptographic Exchanges and Smart Contracts

In analyses of the threat vectors facing smart contracts and their applications, source code, and exchanges, Kudelski begins with a testing regimen that follows a best-practices heuristic developed out of overarching industry standards developed by the OWASP Top 10 and CWE-Mitre's Top 25 Most Dangerous Software Weaknesses. To ensure testing standards address common weaknesses in decentralized cryptocurrency exchanges and smart contracts, Kudelski pays special attention to vulnerabilities highlighted in the DASP-Top 10 (Decentralized Application Security Project – dasp.co), Known Attacks enumerated by ConsenSys's Ethereum Smart Contract Best Practices (https://consensys.github.io/smart-contract-best-practices/known_attacks/) and in consideration of the Smart Contract Weakness Classification Registry (https://swcregistry.io/).

Reporting

To finalize the assessment activity, Kudelski documents the assessment vulnerabilities, endpoints, and findings in a report that summarizes the results into actionable items for remediation by the Client. Each finding documents the steps required to reproduce identified vulnerabilities and includes recommendations for remediating or mitigating the threat.

Tools

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

- Visual Studio Code - https://code.visualstudio.com
- Semgrep - https://semgrep.dev
- Dependency Check
- Cargo Audit

Vulnerability Scoring Systems

Kudelski Security utilizes commonly available vulnerability scoring systems and taxonomies to identify and assign a risk severity to findings.

- Common Vulnerability Scoring System (CVSS)
- Common Weakness Enumeration (CWE)
- Open Web Application Security Project (OWASP)

CWE

The CWE system is a community-developed list of common software security weaknesses. It serves as a common language, a measuring stick for software security tools, and as a baseline for weakness identification, mitigation, and prevention efforts. Some common types of software weaknesses classified by the CWE are:

- Buffer Overflows, Format Strings, etc.
- Structure and Validity Problems
- Common Special Element Manipulations
- Channel and Path Errors
- Handler Errors
- User Interface Errors
- Pathname Traversal and Equivalence Errors
- Authentication Errors
- Resource Management Errors
- Insufficient Verification of Data
- Code Evaluation and Injection
- Randomness and Predictability

KUDELSKI SECURITY CONTACTS

NAME	POSITION	CONTACT INFORMATION