deBridge - Solana Contracts

Solana Program Security Audit

Prepared by: **Halborn**

Date of Engagement: **April 25th, 2022 - May 25th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	05/24/2022	Mateusz Garncarek
0.2	Document Edit	05/25/2022	Mateusz Garncarek
0.3	Final Draft	05/25/2022	Mateusz Garncarek
0.4	Draft Review	05/27/2022	Gabi Urrutia
1.0	Remediation Plan	06/06/2022	Mateusz Garncarek
1.1	Remediation Plan Review	06/07/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Mateusz Garncarek	Halborn	Mateusz.Garncarek@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

deBridge is a cross-chain interoperability and liquidity transfer protocol that allows decentralized transfer of assets between various blockchains. The cross-chain intercommunication of deBridge programs is powered by the network of independent oracles/validators which are elected by deBridge governance.

deBridge engaged Halborn to conduct a security assessment on their Solana programs beginning on April 25th, 2022 and ending May 25th, 2022. The security assessment was scoped to the Solana programs provided in the debridge-finance/solana-contracts GitHub repository.

1.2 AUDIT SUMMARY

The team at Halborn was provided four weeks for the engagement and assigned one full-time security engineer to audit the security of the Solana programs. The security engineer is a blockchain, smart contract and Solana program security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit to achieve the following:

- Ensure that Solana programs functions are intended.
- Identify potential security issues with the Solana programs.

In summary, Halborn identified some security risks that were addressed and acknowledged by the deBridge team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual review of the code and automated security testing to balance efficiency, timeliness, practicality, and

accuracy in regard to the scope of the Solana program audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of Solana programs and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture, purpose, and use of the platform.
- Manual code review and walkthrough.
- Manual assessment of use and safety for the critical Rust variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual assessment to determine access control issues such as missing ownership checks, missing signer checks, and Solana account confusions.
- Fuzz testing. (Halborn custom fuzzing tool).
- Scanning of Rust files for vulnerabilities (cargo audit).
- Detecting usage of unsafe Rust code (cargo-geiger).
- Scanning for common Solana vulnerabilities (soteria).

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.

2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

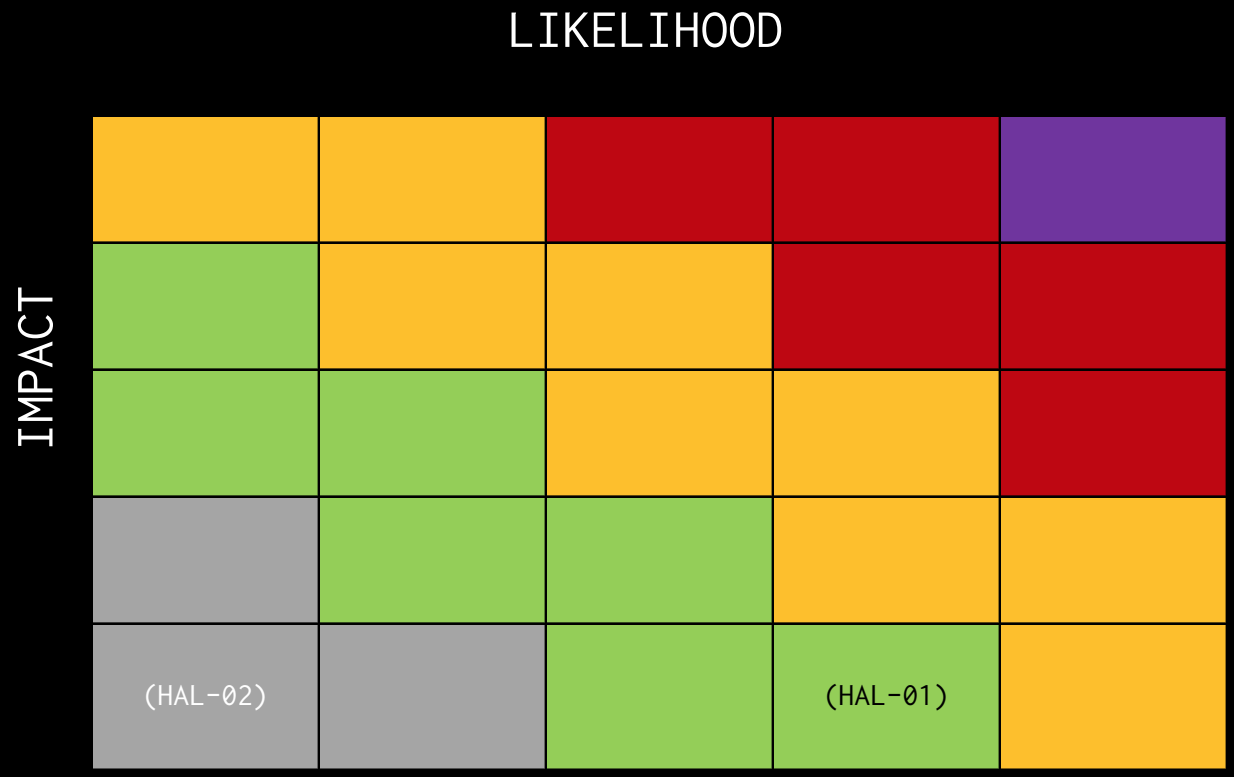
1. Repository: debridge-finance/solana-contracts
2. Commit: 7aeb5dfadbdb1bb049a96a987655723c76c8d0a7
3. Programs in-scope:
   (a) debridge
   (b) settings

OUT-OF-SCOPE:

- Other Solana programs in the repository
- Economics attacks
- Third-party dependencies

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	1	1



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) USAGE OF VULNERABLE CRATES	Low	RISK ACCEPTED
(HAL-02) MISSING PROPER ERROR HANDLING	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) USAGE OF VULNERABLE CRATES - LOW

Description:

It was observed that the project uses crates with known vulnerabilities.

Code Location:

ID	package	short description
RUSTSEC-2020-0036	failure	unmaintained

Risk Level:

Likelihood - 4
Impact - 1

Recommendation:

Even if those vulnerable crates cannot impact the underlying application, it is advised to be aware of them. Also, it is necessary to set up dependency monitoring to always be alerted when a new vulnerability is disclosed in one of the project crates.

Remediation Plan:

RISK ACCEPTED: The DeBridge team accepted the risk of this finding.

3.2 (HAL-02) MISSING PROPER ERROR HANDLING - INFORMATIONAL

Description:

The function deserialize() is using unwrap() methods which are resulting in T but if instead there will be E or None result, then it will panic.

Code Location:

Listing 1: crates/signature-verifier/src/lib.rs (Lines 229,231)

```
205 impl<'instr> RecoverInstruction<'instr> {
206     pub fn deserialize(buf: &mut &'instr [u8], ix_index: u8) -> io
 ↳ ::Result<Self> {
207         const SERIALIZED_SIZE: usize = 11;
208 
209         let count = buf[0] as usize;
210         let offsets: Vec<SecpSignatureOffsets> = {
211             let mut buf: &[u8] = &buf[1..count * SERIALIZED_SIZE +
 ↳  1];
212             (0..count)
213                 .map(|_| {
214                     let offset = bincode::deserialize::<
 ↳ SecpSignatureOffsets>(buf)
215                         .map_err(|err| io::Error::new(io::
 ↳ ErrorKind::InvalidInput, err))?;
216                     buf = &buf[SERIALIZED_SIZE..];
217                     Ok(offset)
218                 })
219                 .take(count)
220                 .collect::<io::Result<Vec<_>>>()
221         }?;
222         Ok(Self {
223             signatures: offsets
224                 .into_iter()
225                 .filter(|offset| offset.message_instruction_index.
 ↳ eq(&ix_index))
226                 .filter(|offset| offset.
 ↳ eth_address_instruction_index.eq(&ix_index))
```

```
227                 .filter(|offset| offset.
 ↳ signature_instruction_index.eq(&ix_index))
228                 .map(|offset| SignatureContext::<'instr> {
229                     address: buf[offset.get_eth_address_range()].
 ↳ try_into().unwrap(),
230                     msg: &buf[offset.get_message_range()],
231                     signature: buf[offset.get_signature_range()].
 ↳ try_into().unwrap(),
232                     recovery_id: &buf[offset.
 ↳ get_recovery_id_offset()],
233                 })
234                 .collect::<Vec<_>>(),
235         })
236     }
237 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to change all unwrap() methods to unwrap_or() or ? in production environment.

Remediation Plan:

ACKNOWLEDGED: The DeBridge team acknowledged this finding.

AUTOMATED TESTING

4.1 VULNERABILITIES AUTOMATIC DETECTION

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and vulnerabilities. Among the tools used were cargo-audit and Soteria. All vulnerabilities published in https://crates.io are stored in a repository named The RustSec Advisory Database. cargo audit is a human-readable version of the advisory database which performs a scanning on Cargo.lock. Security Detections are only in scope. All vulnerabilities shown here were already disclosed in the above report. However, to better assist the developers maintaining this code, the auditors are including the output with the dependencies tree, and this is included in the cargo audit output to better know the dependencies affected by unmaintained and vulnerable crates. Soteria performed a scan on all the programs and sent the compiled results to the analyzers to locate any vulnerabilities.

cargo-audit results:

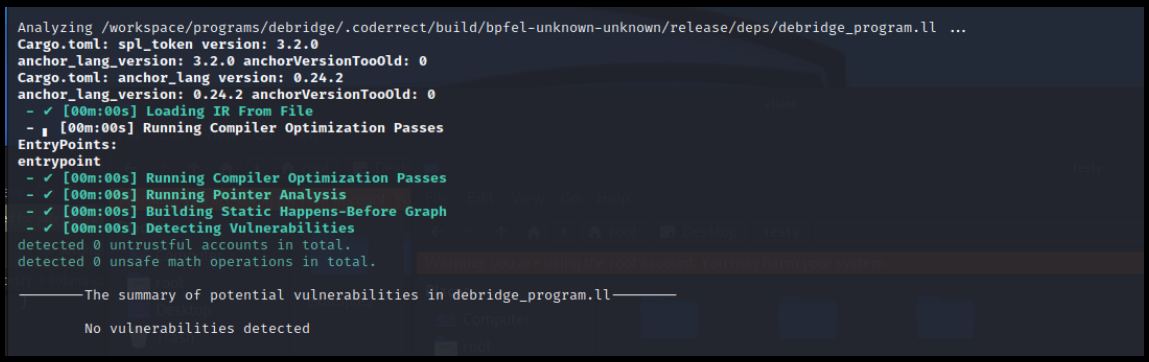
ID	package	short description
RUSTSEC-2020-0036	failure	unmaintained

4.2 AUTOMATED VULNERABILITY SCANNING

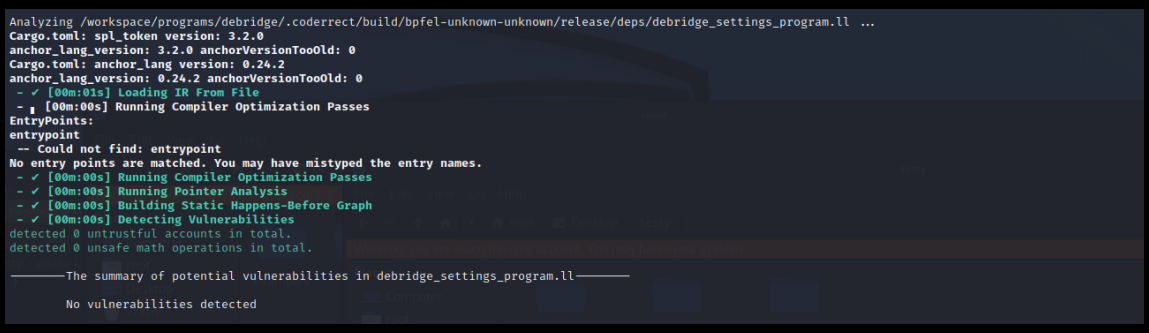
Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruits on the targets for this engagement. Among the tools used was Soteria, a security analysis service for Solana programs. Soteria had scanned the programs in scope and sent the compiled results to the analyzers to look for vulnerabilities.

programs/debridge



programs/settings



4.3 UNSAFE RUST CODE DETECTION

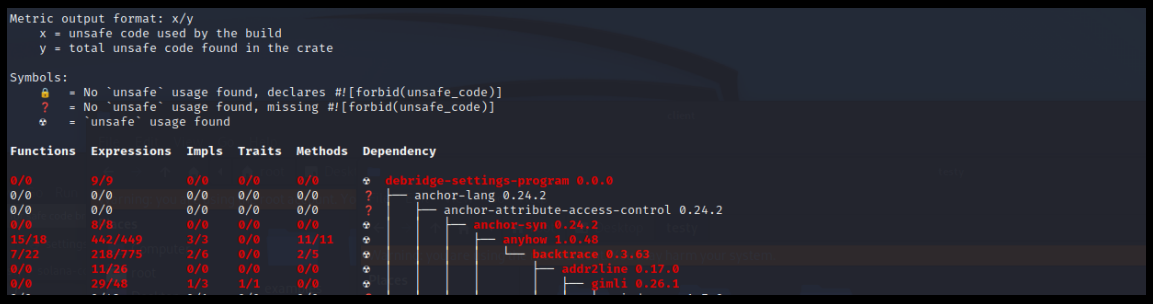
Description:

Halborn used automated security scanners to assist with the detection of well-known security issues and vulnerabilities. Among the tools used was cargo-geiger, a security tool that lists statistics related to the usage of unsafe Rust code in a core Rust codebase and all its dependencies.

Results:

debridge

settings



THANK YOU FOR CHOOSING

// HALBORN