HALBORN

Debridge - solana-tx-parser

Whitebox Pentest

Prepared by: **Halborn**

Date of Engagement: **August 5th, 2022 - August 26th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	08/26/2022	Mateusz Garncarek
0.2	Final Draft	08/26/2022	Mateusz Garncarek
0.3	Draft Review	08/26/2022	Piotr Cielas
0.4	Draft Review	08/26/2022	Gabi Urrutia
1.0	Remediation Plan	09/13/2022	Przemyslaw Swiatowiec
1.1	Remediation Plan Review	09/13/2022	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
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com

Przemyslaw Swiatowiec	Halborn	Przemyslaw.Swiatowiec@halborn.com
Mateusz Garncarek	Halborn	Mateusz.Garncarek@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Debridge engaged Halborn to conduct a security audit on their solana-tx-parser library, beginning on August 5th, 2022 and ending on August 26th, 2022 . Halborn was provided access to library's source code to conduct whitebox security testing using tools to scan, detect and validate possible vulnerabilities found in the application and report the findings at the end of the engagement.

solana-tx-parser is a Typescript library which provides features like instruction deserialization, transaction flattening and transaction log messages parsing. The library is open source and those functions can be used for debugging and reading the logs in any Solana project.

The security assessment was scoped to the programs provided in the solana-tx-parser-public GitHub repository. Commit hashes and further details can be found in the **Scope** section of this report.

1.2 AUDIT SUMMARY

The team at Halborn was provided three weeks for the engagement and assigned three full-time security engineers to audit the security of the assets in scope. The engineers are blockchain and smart contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The goals of our security audits are to improve the quality of systems by testing this as whitebox approach. We review and aim for sufficient remediation to help protect users.

In summary, Halborn identified some improvements to reduce the likelihood and impact of multiple risks that were successfully addressed by the DeBridge team:

- logs with truncated messages are parsed appropriately,
- log messages with multiple lines are parsed correctly.

1.3 TEST APPROACH & METHODOLOGY

Halborn applied the Whitebox methodology and performed a combination of manual and automated security testing in order to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the pentest. While manual testing is recommended to uncover flaws in logic, process and implementation; automated testing techniques assist enhance coverage of the infrastructure and can quickly identify flaws in it. The following phases and associated tools were used throughout the term of the audit:

- Research into the architecture, purpose, and use of the program.
- Mapping Application Content and Functionality
- Application Logic Flaws
- Input Handling
- Fuzzing of all input parameters
- Scanning dependencies for known vulnerabilities (npm-audit)
- Performing static analysis (sonarqube)

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

The following Git repositories were in scope:

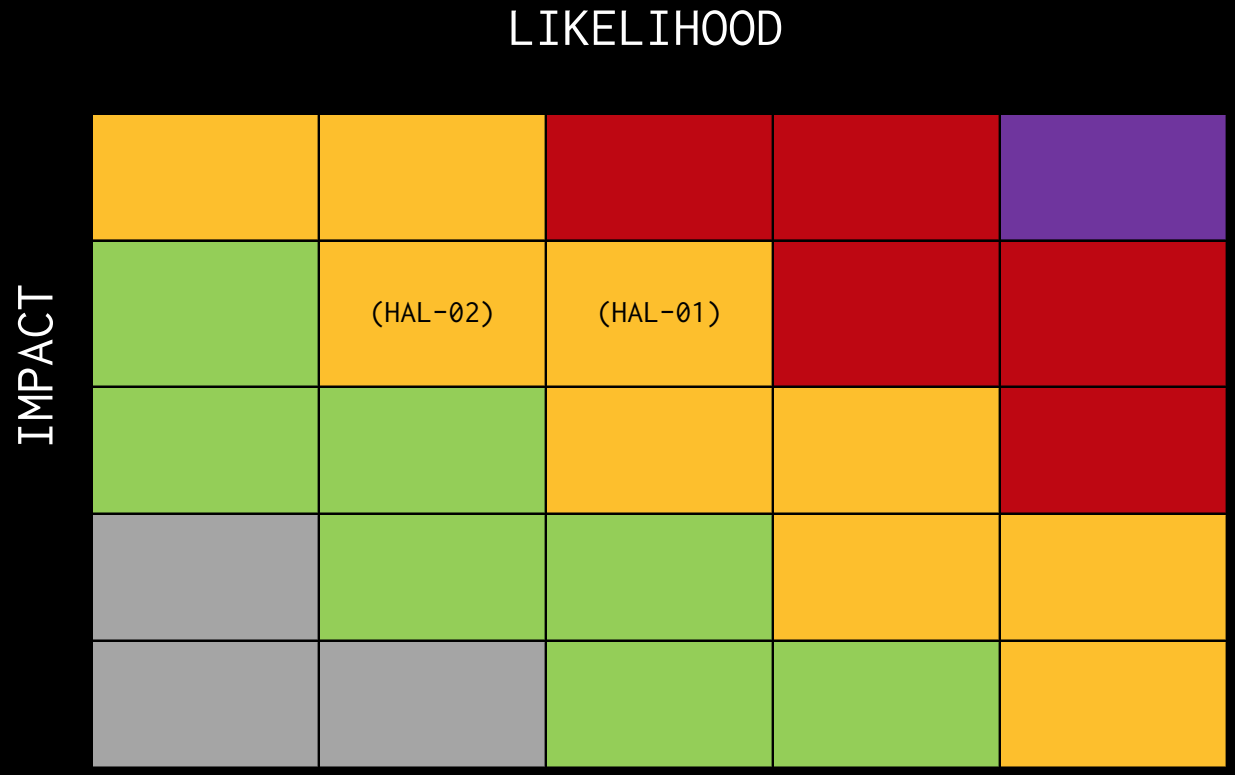
1. solana-tx-parser-public

   (a) Repository: solana-tx-parser-public

   (b) Commit ID: e20c4c4f1690e05a25a6d91b54d4f8757c8b4ec2

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	2	0	0



EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - LOGS WITH NEWLINE CANNOT BE PARSED	Medium	SOLVED - 08/26/2022
HAL-02 - TRUNCATED LOGS CANNOT BE PARSED	Medium	SOLVED - 08/26/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) LOGS WITH NEWLINE CANNOT BE PARSED - MEDIUM

Description:

Solana supports log messages with newline character `\n`. However, it was observed that `solana-tx-parser` cannot parse such logs. If a user sends a transaction to a program that emits a log message in which there is a new line character, the library will reject such transaction and will not parse it, even if all following logs contain only flat messages without newlines.

Code location:

Listing 1: solana-tx-parser/src/helpers.ts (Line 161)

```
160 if (!match || !match.groups) {
161     throw new Error(`Failed to parse log line: ${log}`);
162 }
```

Risk Level:

Likelihood - 3
Impact - 4

Proof Of Concept:

1. Create a program with an instruction handler that contains the following code:

Listing 2: (Line 12)

```
11 pub fn some_instruction(ctx: Context<Context>) -> Result<()> {
12     msg!("Log with new line\n");
13     Ok(())
14 }
```

2. Deploy the program and send a transaction targeting that handler. Observe that the instruction is executed correctly and the following log message was emitted:

Listing 3: (Line 17)

```
14 Program ChtRDTGjhUYubbavybASVJv3h7ZsfDzURvPFemgfRbcb invoke [1]
15     Program log: Instruction: InitializeSendBridge
16     Program log: Log with new line
17
18     Program ChtRDTGjhUYubbavybASVJv3h7ZsfDzURvPFemgfRbcb consumed
↳ 5226 of 200000 compute units
19     Program ChtRDTGjhUYubbavybASVJv3h7ZsfDzURvPFemgfRbcb success
```

3. Try to parse the transaction with solana-tx-parser and observe that the parser throws an error.

Recommendation:

It is recommended to gracefully handle parsing transactions with log messages that span over multiple lines and which including the newline character.

Remediation Plan:

SOLVED: The DeBridge team fixed this issue by changing the regex parsing to accept multiline log messages in commit 63fdd1afbae5f0ae2788e0f77ce5616d05415f54.

3.2 (HAL-02) TRUNCATED LOGS CANNOT BE PARSED - MEDIUM

Description:

Solana truncates log messages that are too long. Such logs appear as `Log truncated` in transaction log trail. However, it was observed that `solana-tx-parser` cannot parse such logs.

If a user supplies a transaction to a program that emits a truncated log message, such transaction will be rejected and will not be able to be parsed by the library.

Code location:

Listing 4: solana-tx-parser/src/helpers.ts (Line 161)

```
160 if (!match || !match.groups) {
161     throw new Error(`Failed to parse log line: ${log}`);
162 }
```

Listing 5: solana-tx-parser/src/helpers.ts (Line 151)

```
150 const parserRe =
151     /(?<programInvoke>^Program (?<invokeProgramId>[1-9A-HJ-NP-Za-
 ↳ km-z]{32,}) invoke \[(?<level>\d+)\]$)|(?<programSuccessResult>^
 ↳ Program (?<successResultProgramId>[1-9A-HJ-NP-Za-km-z]{32,})
 ↳ success$)|(?<programFailedResult>^Program (?<failedResultProgramId
 ↳ >[1-9A-HJ-NP-Za-km-z]{32,}) failed: (?<failedResultErr>.*)$)|(?<
 ↳ programCompleteFailedResult>^Program failed to complete: (?<
 ↳ failedCompleteError>.*)$)|(?<programLog>^^Program log: (?<
 ↳ logMessage>.*)$)|(?<programData>^Program data: (?<data>.*)$)|(?<
 ↳ programConsumed>^Program (?<consumedProgramId>[1-9A-HJ-NP-Za-km-z
 ↳ ]{32,}) consumed (?<consumedComputeUnits>\d*) of (?<
 ↳ allComputedUnits>\d*) compute units$)|(?<programReturn>^Program
 ↳ return: (?<returnProgramId>[1-9A-HJ-NP-Za-km-z]{32,}) (?<
 ↳ returnMessage>.*)$)/;
152 const result: LogContext[] = [];
```

Proof Of Concept:

1. Create a program with instruction handler that contains the following code:

Listing 6: (Line 12)

```
11 pub fn some_instruction(ctx: Context<Context>) -> Result<()> {
12     msg!("{PUT LONG STRING HERE, AT LEAST 10 000 characters}");
13     Ok(())
14 }
```

2. Deploy the program and send a transaction targeting that handler. Observe that the instruction is correctly executed, and the following log message was emitted:

Listing 7: (Line 17)

```
14   Log Messages:
15     Program ChtRDTGjhUYubbavybASVJv3h7ZsfDzURvPFemgfRbcb invoke
↳ [1]
16     Program log: Instruction: InitializeSendBridge
17     Log truncated
18     Program ChtRDTGjhUYubbavybASVJv3h7ZsfDzURvPFemgfRbcb consumed
↳ 18594 of 200000 compute units
19     Program ChtRDTGjhUYubbavybASVJv3h7ZsfDzURvPFemgfRbcb success
```

3. Try to parse the transaction with solana-tx-parser and observe that the parser throws an error.

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

It is recommended to gracefully handle parsing transactions with log messages that are truncated by the Solana runtime.

Remediation Plan:

SOLVED: The DeBridge team fixed this issue by changing parsing logic to handle truncated logs in commit 63fdd1afbae5f0ae2788e0f77ce5616d05415f54.

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 npm audit. All vulnerabilities published in https://npmjs.com/advisories are stored in an advisories entries. npm audit is checking configured dependencies for project and asks for a report of known vulnerabilities found for them in that advisories. Security Detections are only in scope. All vulnerabilities shown here were already disclosed in the above report.

Results:

No security issues were detected.

```
root@[redacted]:~/solana-tx-parser-master (1)/solana-tx-parser-master# npm audit
found 0 vulnerabilities
```

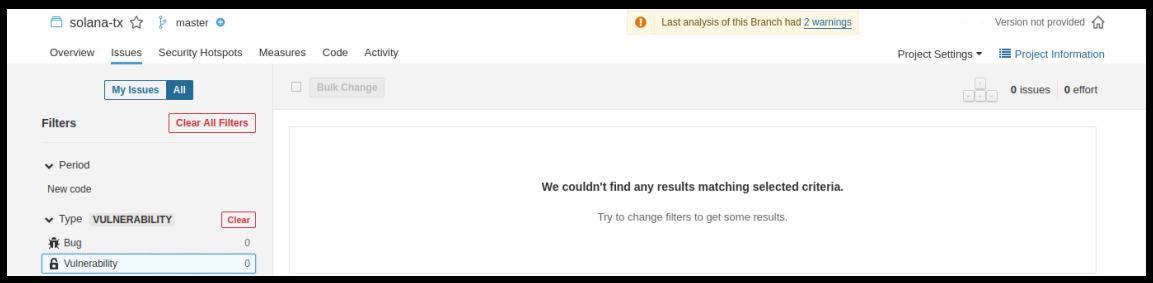
4.2 STATIC ANALYSIS

Description:

Halborn used static security code 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 sonarqube, a static security code scanner for JavaScript programs. Sonarqube had scanned the library in scope and sent the compiled results to the analyzers to look for vulnerabilities.

Results Sonarqube:

No security issues were detected.



THANK YOU FOR CHOOSING

// HALBORN