// HALBORN

Debridge - Solana Event Reader

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	08/19/2022	Przemyslaw Swiatowiec
0.2	Document Edits	08/26/2022	Przemyslaw Swiatowiec
0.3	Final Draft	08/26/2022	Przemyslaw Swiatowiec
0.4	Draft Review	08/26/2022	Piotr Cielas
0.5	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

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 Event Reader application, beginning on August 5th, 2022 and ending on August 26th, 2022 . Halborn was provided access to the program's source code to conduct whitebox security testing using tools to scan, detect, validate possible vulnerabilities found in the application and report the findings at the end of the engagement.

Solana Event Reader is an application that exports API to handle four types of Debridge events - Sent (Transferred), Claimed (Bridged), SendBridgeInitialized, MintBridgeInitialized. The application uses both RPC and Websocket protocols to communicate with the provided Solana node to retrieve transaction logs. Those logs are then parsed with solana-tx-parser library from Solana log format to JSON.

The security assessment was scoped to the programs provided in the solana-events-reader 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 one full-time security engineer to audit the security of the assets in scope. The engineer is a blockchain and smart contract security expert 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 addressed and acknowledged by the DeBridge team.

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 penetration test. 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
- Importing Solana program logs from debridge and settings Solana programs at commit c28bc31309c73cd47d053a1ba91ffa2cb1fec41c
- Input Handling
- Fuzzing of all input parameters
- Scanning dependencies for known vulnerabilities (npm-audit)
- Performing static analysis (nodejsscan, 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.

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

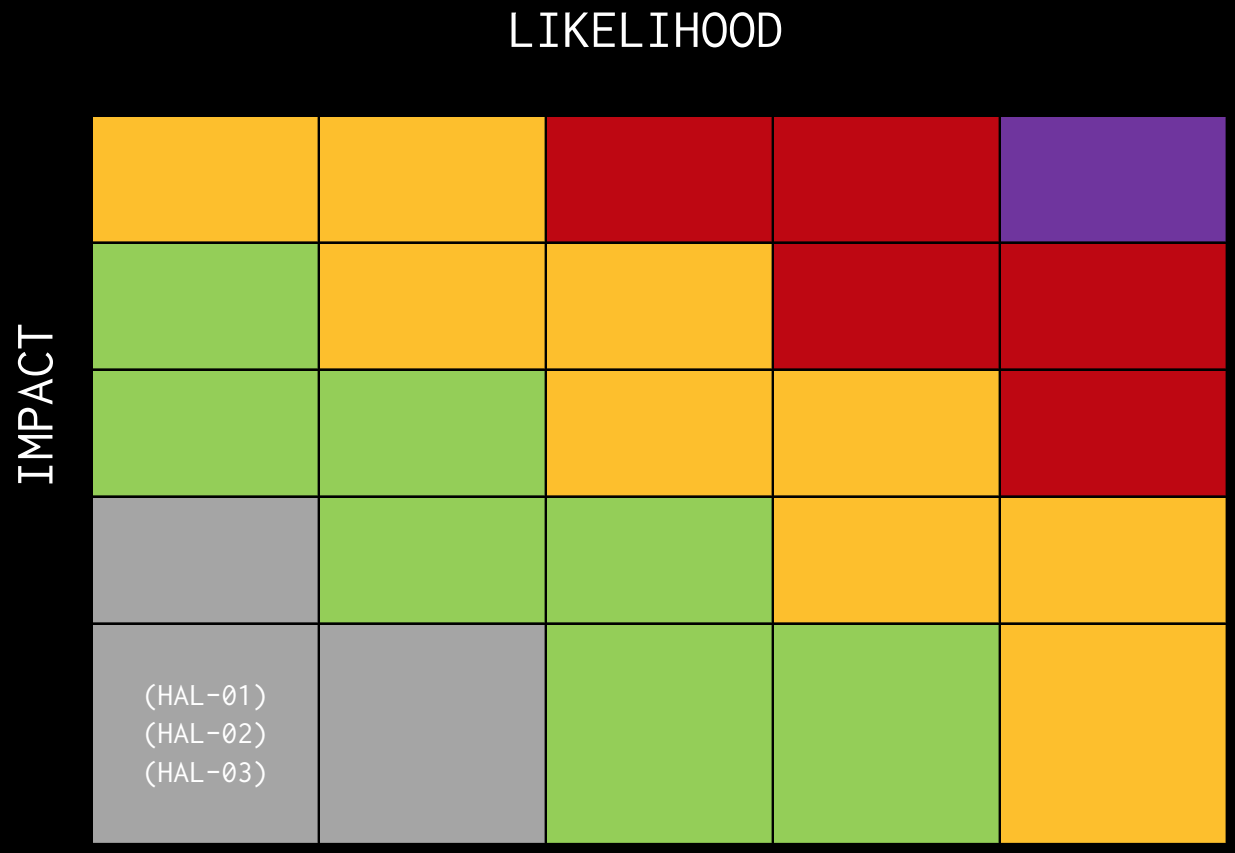
1.4 SCOPE

The following Git repositories are in scope:

1. solana-events-reader
    (a) Repository: solana-events-reader
    (b) Commit ID: ea412051e7826f723ee8aee7cb33295d27ff4254

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	3



EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - APPLICATION PERFORMANCE CONCERNS	Informational	ACKNOWLEDGED
HAL-02 - NO TEST COVERAGE	Informational	ACKNOWLEDGED
HAL-03 - UNREACHABLE CODE	Informational	SOLVED - 08/30/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) APPLICATION PERFORMANCE CONCERNS - INFORMATIONAL

Description:

During application dynamic testing, several performance concerns were raised:

- **Application should only be used with private Solana node**
  Running the application with public nodes can result in IP blacklisting due to hitting request rate limitations. To avoid reputational damage due to blacklisting, it's recommended to use only dedicated private nodes.
- **Application API should be protected with rate limiter**
  Solana Event Reader application could be treated as an endpoint to a private Solana node. Users can invoke many API requests like `POST getEventsFromTransactions` to decrease performance of the node connected to the application. As rate-limiting is not implemented in the code, it's recommended to introduce a rate-limiting component in application deployment infrastructure.
- **Possibility to send computation expensive requests**
  It was observed that following API endpoints could be used to decrease application performance:
  - `POST getBridgedEventsFromTransactions`
  - `POST getBridgeInitializedEventsFromTransactions`
  - `POST getBridgeInitializedEventsFromTransactions`
  - `POST getEventsFromTransactions`

The endpoints listed earlier accept up to 200 transaction signatures in one request. Malicious users can flood the application with such requests, and request the maximum number of transactions signatures. Even with a

rate-limiting solution in-place, the application has to perform cost-expensive tasks to pull and parse 200 transactions. It's recommended either to decrease the number of signatures available per request, or introduce an authorization layer, so that only authorized applications / users (in ex. with API key) can request more transactions in one batch. What is more, introducing a caching layer could be worth to consider. With a caching layer, instead of pulling and parsing a transaction, the application could get ab already parsed transaction from cache storage.

Code Location:

Listing 1: request/get.bridge.initialized.events.from.transactions.request.dto.ts (Line 9)

```
 4 export class GetBridgeInitializedEventsFromTransactionsRequestDto
↳ {
 5   @ApiProperty()
 6   @IsArray()
 7   @ArrayMinSize(1)
 8   @ArrayMaxSize(200)
 9   signatures: string[];
10 }
11
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to:

- introduce caching to the transaction parsing mechanism.
- allow the application only be used with private Solana nodes.
- protect the application API with a rate-limiting component to minimize the risk of issues with private node performance.

Remediation Plan:

ACKNOWLEDGED: The \client team acknowledged this finding. As Solana Event Reader will be used as an isolated product in a closed container, the \client team decided that it is not necessary to introduce a rate-limiting and authorization layer.

3.2 (HAL-02) NO TEST COVERAGE - INFORMATIONAL

Description:

It was observed that Solana Events Reader project is missing both unit and e2e tests. The repository contains only one e2e and one unit test to check if the application could be started without errors. There are no tests to verify if API endpoints are working as expected or if transactions are parsed correctly.

Unit tests and functional testing are recommended to ensure the code works correctly before deployment. Missing tests, it is easier to introduce regression bugs, also security, and broken functionality issues.

Code Location:

Listing 2: solana-events-reader/src/app.controller.spec.ts (Line 5)

```
5  describe('AppController', () => {
6    let appController: AppController;
7
8    beforeEach(async () => {
9      const app: TestingModule = await Test.createTestingModule({
10       controllers: [AppController],
11       providers: [AppService],
12     }).compile();
13
14     appController = app.get<AppController>(AppController);
15   });
16
17   describe('root', () => {
18     it('should return "Hello World!"', () => {
19       expect(appController.getHello()).toBe('Hello World!');
20     });
21   });
22 });
23
```

Listing 3: solana-events-reader/test/app.e2e-spec.ts (Line 6)

```
6  describe('AppController (e2e)', () => {
7    let app: INestApplication;
8
9    beforeEach(async () => {
10     const moduleFixture: TestingModule = await Test.
↳ createTestingModule({
11       imports: [AppModule],
12     }).compile();
13
14     app = moduleFixture.createNestApplication();
15     await app.init();
16   });
17
18   it('/ (GET)', () => {
19     return request(app.getHttpServer())
20       .get('/')
21       .expect(200)
22       .expect('Hello World!');
23   });
24 });
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to introduce both e2e and unit tests that perform as many test cases as possible to cover all conceivable scenarios in the application.

Remediation Plan:

ACKNOWLEDGED: The Debridge team acknowledged this finding.

3.3 (HAL-03) UNREACHABLE CODE - INFORMATIONAL

Description:

The Solana Events Reader contains dead code, which can never be executed. Jump statements (return, break and continue) and throw expressions move control flow out of the current code block. So, any statements that come after a jump are dead code.

Existing of unreachable code decreases readability of code-base and unnecessary increases application size.

Code Location:

Listing 4: solana-events-reader/src/services/get-token-decimals.service.ts (Lines 38-39)

```
34 return rawMint.decimals;
35 /*
36 Or just use existing function
37 */
38 const mint = await getMint(connection, tokenMint);
39 return mint.decimals;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to remove unreachable code.

Remediation Plan:

SOLVED: The Debridge team fixed this issue by removing the unreachable code in commit bbca68f5857f9736ac4215cdce2216ced5955338

AUTOMATED TESTING

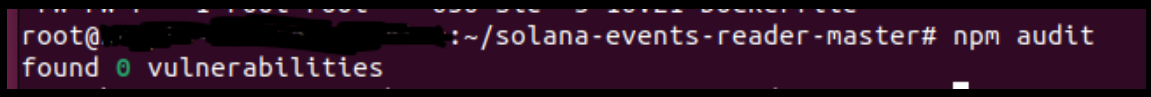
4.1 VULNERABILITIES AUTOMATIC DETECTION

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.



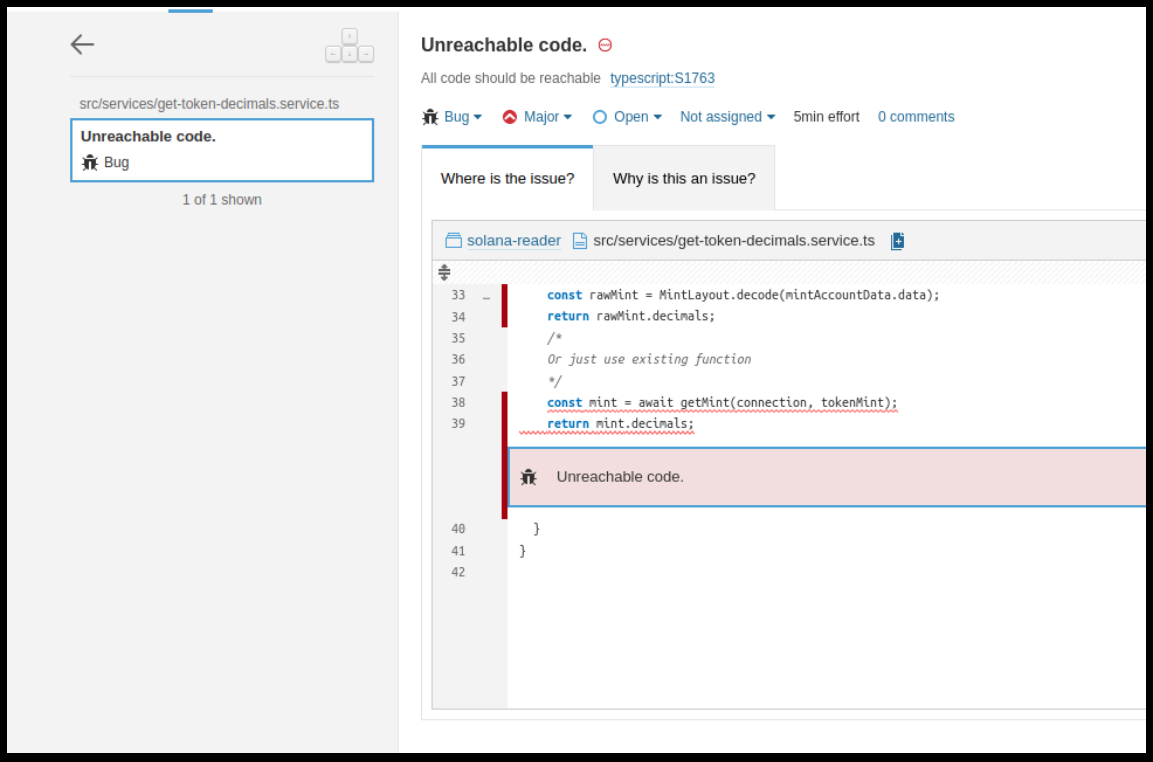
4.2 AUTOMATED VULNERABILITY SCANNING

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 were nodejsscan and sonarqube, static security code scanners for javascript programs. Nodejsscan and sonarqube had scanned an application in the scope and sent the compiled results to the analyzers to locate any vulnerabilities.

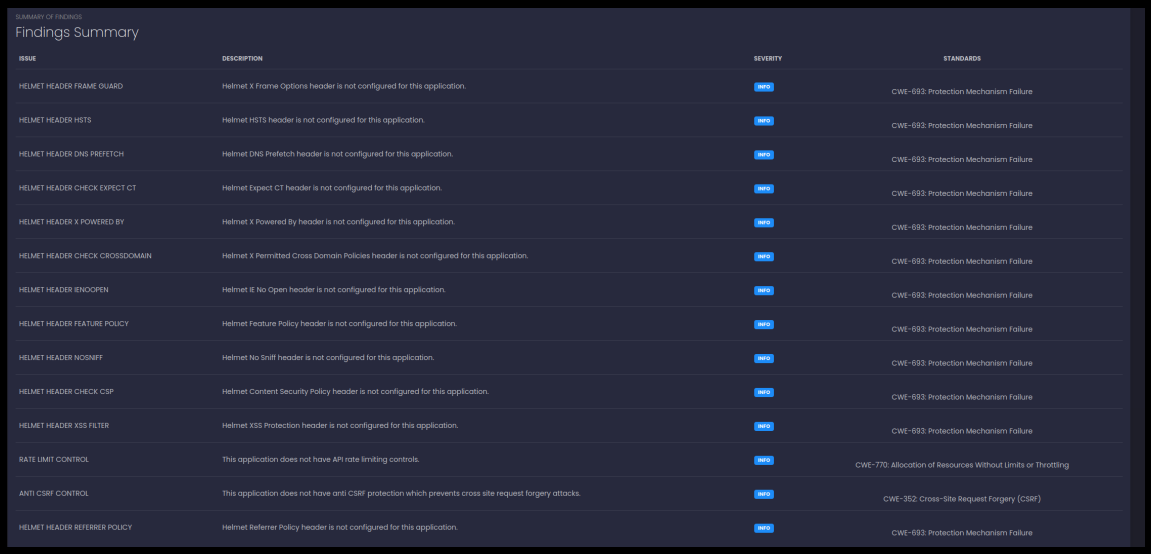
Results Sonarqube:

Sonarqube reported one bug - unreachable code, which was reported as HAL-02 in the previous report section.



Results Nodejsscan:

Nodejsscan reported some missing HTTP headers issues. However, these were treated as false-positive and excluded from the report as they are out-of scope source code / white-box audit.



SUMMARY OF FINDINGS

Findings Summary

ISSUE	DESCRIPTION	SEVERITY	STANDARDS
HELMET HEADER FRAME GUARD	Helmet X Frame Options header is not configured for this application.	INFO	CWE-693: Protection Mechanism Failure
HELMET HEADER HSTS	Helmet HSTS header is not configured for this application.	INFO	CWE-693: Protection Mechanism Failure
HELMET HEADER DNS PREFETCH	Helmet DNS Prefetch header is not configured for this application.	INFO	CWE-693: Protection Mechanism Failure
HELMET HEADER CHECK EXPECT CT	Helmet Expect CT header is not configured for this application.	INFO	CWE-693: Protection Mechanism Failure
HELMET HEADER X POWERED BY	Helmet X Powered By header is not configured for this application.	INFO	CWE-693: Protection Mechanism Failure
HELMET HEADER CHECK CROSSDOMAIN	Helmet X Permitted Cross Domain Policies header is not configured for this application.	INFO	CWE-693: Protection Mechanism Failure
HELMET HEADER IENOOPEN	Helmet IE No Open header is not configured for this application.	INFO	CWE-693: Protection Mechanism Failure
HELMET HEADER FEATURE POLICY	Helmet Feature Policy header is not configured for this application.	INFO	CWE-693: Protection Mechanism Failure
HELMET HEADER NOSNIFF	Helmet No Sniff header is not configured for this application.	INFO	CWE-693: Protection Mechanism Failure
HELMET HEADER CHECK CSP	Helmet Content Security Policy header is not configured for this application.	INFO	CWE-693: Protection Mechanism Failure
HELMET HEADER XSS FILTER	Helmet XSS Protection header is not configured for this application.	INFO	CWE-693: Protection Mechanism Failure
RATE LIMIT CONTROL	This application does not have API rate limiting controls.	INFO	CWE-770: Allocation of Resources Without Limits or Throttling
ANTI CSRF CONTROL	This application does not have anti CSRF protection which prevents cross site request forgery attacks.	INFO	CWE-352: Cross-Site Request Forgery (CSRF)
HELMET HEADER REFERRER POLICY	Helmet Referrer Policy header is not configured for this application.	INFO	CWE-693: Protection Mechanism Failure

THANK YOU FOR CHOOSING

// HALBORN