DeBridge - DLN Taker

Code Security Assessment

Prepared by: **Halborn**

Date of Engagement: **August 18th, 2023 - September 15th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	08/18/2023
0.2	Document Edits	09/15/2023
0.3	Draft Review	09/18/2023
1.0	Remediation Plan	09/25/2023
1.1	Remediation Plan Review	09/25/2023

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
Erlantz Saenz	Halborn	Erlantz.Saenz@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

DeBridge engaged Halborn to conduct a security assessment on the dln take code beginning on August 18th, 2023 and ending on September 15th, 2023. The security assessment was scoped to the code provided to the Halborn team.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided three weeks for the engagement and assigned a full-time security engineer to verify the security of the dln take code. The security engineer is a blockchain and smart contract security expert with advanced penetration testing, smart contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that code functions operate as intended
- Identify potential security issues within the application code

In summary, Halborn did not identify any security issue that could expose the integrity of the application.
The code evaluated presented a limited attack surface. Additionally, all the confidential data was correctly placed, which reduced the surface to a local user with access to the file.
Halborn, performed a code review evaluating different points where the application could be attacked, however no points of failure were found.

1.3 SCOPE

dln-taker is a rule-based daemon service built to automatically execute profitable orders placed on the deSwap Liquidity Network (DLN) across supported blockchains.

DLN-TAKER
- Commit ID: 81f2e304ed900ad2b4e60b2c83e7f0f2be7bddb2

1.4 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process and implementation; automated testing techniques help enhance coverage of the code 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 assessment:

- Mapping Application Content and Functionality
- Technology stack-specific vulnerabilities and Code Assessment
- Known vulnerabilities in 3rd party / OSS dependencies
- Application Logic Flaws
- Authentication / Authorization flaws
- Input Handling
- Fuzzing of all input parameters
- Testing for different types of sensitive information leakages: memory, clipboard, etc.
- Test for Injection (SQL/JSON/HTML/JS/Command/Directories. . .)
- Brute Force Attempts
- Perform static analysis on code
- Ensure that coding best practices are being followed by DeBridge team
- Technology stack-specific vulnerabilities and Code Assessment
- Known vulnerabilities in 3rd party / OSS dependencies.
- Identify potential vulnerabilities that may pose a risk to DeBridge

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.

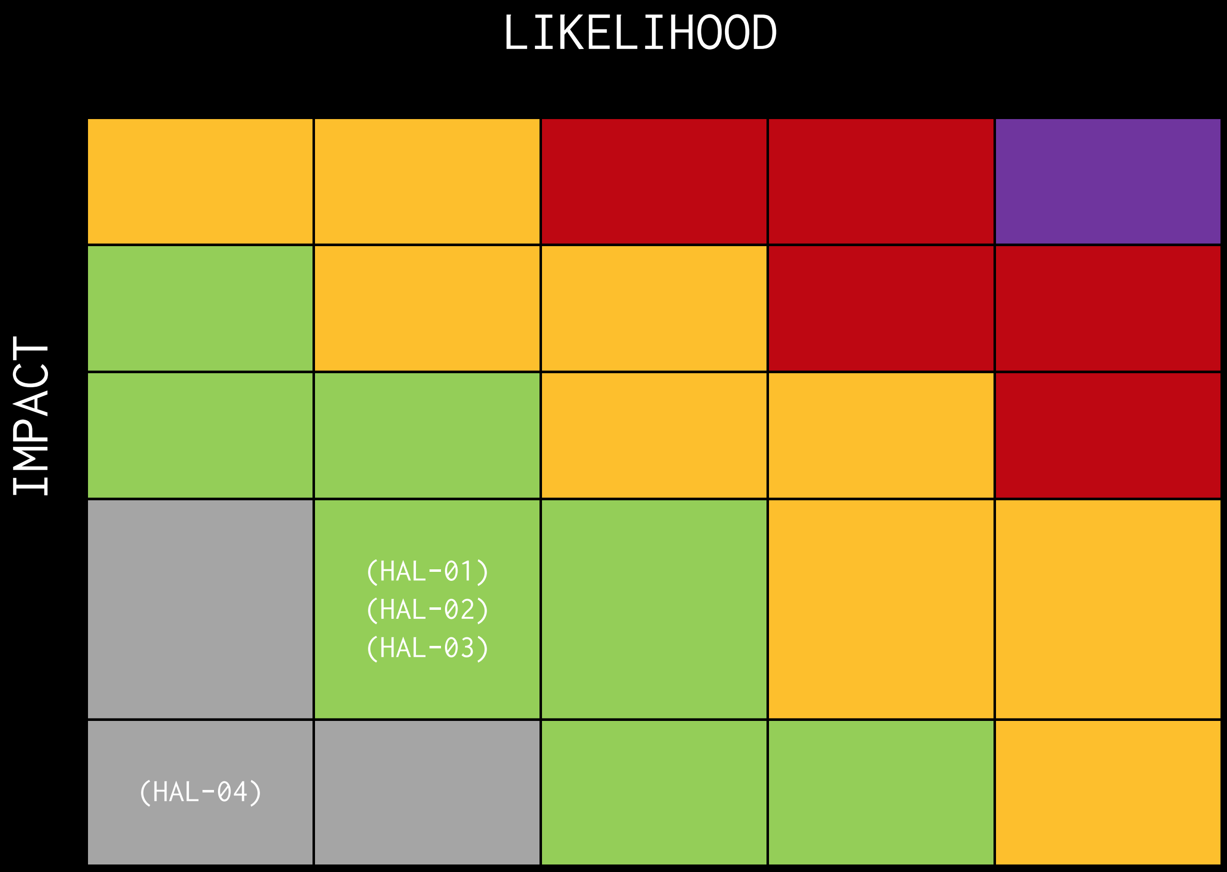
CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - **8** - HIGH
7 - **6** - MEDIUM
5 - **4** - LOW

3 - 1 - VERY LOW AND INFORMATIONAL

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	3	1



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) MULTIPLE OUTDATED PACKAGES	Low	RISK ACCEPTED
(HAL-02) LACK OF DEFAULT ON SWITCH	Low	RISK ACCEPTED
(HAL-03) USE OF UNNECESSARY SWITCH STATEMENT	Low	RISK ACCEPTED
(HAL-04) MULTIPLE TODO COMMENTS	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) MULTIPLE OUTDATED PACKAGES - LOW

Description:

During the security assessment, an automated check was performed against the project dependencies. The command yarn audit revealed the use of multiple vulnerable or outdated dependencies.
Disclaimer: During the assessment, not direct impact was found related to the outdated packages.

Evidences:

Package	Vulnerability	Patched version
tough-cookie	tough-cookie Prototype Pollution vulnerability	>=4.1.3
request	Server-Side Request Forgery in Request	-

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Update the dependencies to the last version available.

Remediation Plan:

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

3.2 (HAL-02) LACK OF DEFAULT ON SWITCH - LOW

Description:

The switch statement allows for a series of case checks against a single expression's value. Typically, this structure also allows for a default case, which serves as a fallback when none of the explicitly listed cases match the value of the expression.

The absence of a default case means that the switch structure does not provide a specific course of action when none of the cases match the given value.

Code Location:

Listing 1: src/executors/executor.ts (Lines 213,214)

```
212        for (const chain of config.chains) {
213       switch (getEngineByChainId(chain.chain)) {
214         case ChainEngine.EVM: {
215           addresses[chain.chain] = {
216             pmmSourceAddress:
217               chain.environment?.pmmSrc ||
218               PRODUCTION.defaultEvmAddresses?.pmmSrc ||
219               PRODUCTION.chains[chain.chain]?.pmmSrc,
220             pmmDestinationAddress:
221               chain.environment?.pmmDst ||
222               PRODUCTION.chains[chain.chain]?.pmmDst ||
223               PRODUCTION.defaultEvmAddresses?.pmmDst,
224             deBridgeGateAddress:
225               chain.environment?.deBridgeContract ||
226               PRODUCTION.chains[chain.chain]?.deBridgeContract ||
227               PRODUCTION.defaultEvmAddresses?.deBridgeContract,
228             crossChainForwarderAddress:
229               chain.environment?.evm?.forwarderContract ||
230               PRODUCTION.chains[chain.chain]?.evm?.forwarderContract ||
231               PRODUCTION.defaultEvmAddresses?.evm?.forwarderContract
```

```
            };
          }
        }
      }
```

- src/executors/executor.ts
- src/helpers.ts

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Implement a default case as a fallback when none of the explicitly listed cases match the value of the expression.

Remediation Plan:

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

3.3 (HAL-03) USE OF UNNECESSARY SWITCH STATEMENT - LOW

Description:

Switch statements are used when there are multiple decisions on a program flow. However, when there is only one case, technically is more efficient to use an if statement instead of switch.
Under the hood, the switch statement compares the cases against the switch condition with a Strict Equality Operator (===), which could be replaced by:

Listing 2: Potential code improvement

```
for (const chain of config.chains) {
    if (getEngineByChainId(chain.chain)===ChainEngine.EVM){
        [...REDACTED...]
    }
```

Code Location:

Listing 3: src/executors/executor.ts (Lines 213,214)

```
212       for (const chain of config.chains) {
213      switch (getEngineByChainId(chain.chain)) {
214        case ChainEngine.EVM: {
215          addresses[chain.chain] = {
216            pmmSourceAddress:
217              chain.environment?.pmmSrc ||
218              PRODUCTION.defaultEvmAddresses?.pmmSrc ||
219              PRODUCTION.chains[chain.chain]?.pmmSrc,
220            pmmDestinationAddress:
221              chain.environment?.pmmDst ||
222              PRODUCTION.chains[chain.chain]?.pmmDst ||
223              PRODUCTION.defaultEvmAddresses?.pmmDst,
224            deBridgeGateAddress:
225              chain.environment?.deBridgeContract ||
226              PRODUCTION.chains[chain.chain]?.deBridgeContract ||
227              PRODUCTION.defaultEvmAddresses?.deBridgeContract,
            crossChainForwarderAddress:
              chain.environment?.evm?.forwarderContract ||
              PRODUCTION.chains[chain.chain]?.evm?.forwarderContract ||
              PRODUCTION.defaultEvmAddresses?.evm?.forwarderContract
          };
        }
      }
    }
```

- src/executors/executor.ts

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Evaluate the snippet above, and modify the structure accordingly.

Remediation Plan:

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

3.4 (HAL-04) MULTIPLE TODO COMMENTS - INFORMATIONAL

Description:

Multiple TODO comments were found on the code. From the security perspective, the use of these comments did not imply a security risk. However, it could mean that the developed application did not reach an appropriate level of maturity to be in a production environment.

Code Location:

Listing 4: src/processors/universal.ts (Line 290)

```
289      // forward to the next order
290      // TODO try to get rid of recursion here. Use setInterval?
291      const nextOrderId = this.pickNextOrderId();
292      if (nextOrderId) {
293        this.tryProcess(nextOrderId);
294      }
295    }
```

- src/processors/universal.ts
- src/executors/executor.ts
- src/providers/evm.provider.adapter.ts

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Review all the comments on the code and ensure that this situation does not affect or offer any risk to the security of the application.

Remediation Plan:

ACKNOWLEDGED: The DeBridge team and Halborn agreed that the issue did not present any risk for the application.

THANK YOU FOR CHOOSING

HALBORN