HALBORN

Neonlabs - Proxy Application

WebApp Pentest

Prepared by: **Halborn**

Date of Engagement: **September 1st, 2022 - October 13th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	10/13/2022	Afaq Abid
0.2	Document Edits	10/20/2022	Afaq Abid
0.3	Draft Review	10/20/2022	Constantin Casmir
0.4	Draft Review	10/20/2022	Gabi Urrutia
1.0	Remediation Plan	11/10/2022	Afaq Abid
1.1	Remediation Plan	11/10/2022	George Skouroupathis
1.2	Remediation Plan Review	11/10/2022	Constantin Casmir
1.3	Remediation Plan Review	11/10/2022	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
Afaq Abid	Halborn	Afaq.Abid@halborn.com
George Skouroupathis	Halborn	George.Skouroupathis@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Neonlabs engaged Halborn to conduct a security audit on their Proxy application beginning on September 1st, 2022 and ending on October 13th, 2022. The security assessment was scoped to the smart contracts provided to the Halborn team. Halborn was provided access to the source code of the application, and the testing environment to conduct security testing using tools to scan, detect, validate possible vulnerabilities found in the application and report the findings at the end of the engagement.

1.2 AUDIT SUMMARY

The team at Halborn was provided a timeline for the engagement and assigned two full-time security engineers to audit the security of the assets in scope. Engineers are blockchain and smart contract security expert with advanced penetration testing, smart-contract hacking, web application pentesting, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to achieve the following:

- Ensure that Proxy functions are working as intended.
- Identify potential security issues with the application.

In summary, Halborn identified the risk of SQL injection in one of the proxy application services. Moreover, checks against input in and the use of some insecure functions are missing, unsanitized input reflections were also identified during the assessment. As stated, all reported issues were addressed by Neonlabs.

1.3 TEST APPROACH & METHODOLOGY

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

accuracy in regard to the scope of the program audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of 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:

- Storing private keys and assets securely
- Application Logic Flaws
- Fuzzing of all input parameters
- Areas where insufficient validation allows for hostile input
- Research into architecture and purpose.
- Static Analysis of security for scoped program
- Manual Assessment for discovering security vulnerabilities.
- Known vulnerabilities in 3rd party / OSS dependencies.

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

IN-SCOPE:
The security assessment was scoped to neonlabsorg/proxy-model.py repository.

Commit ID: 56c39293a9e91f59a5cbb5ba6eacb9b727300247

OUT-OF-SCOPE:
External libraries.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	3	0

LIKELIHOOD

IMPACT

	(HAL-01)			
(HAL-02)				
(HAL-03) (HAL-04)				

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
PYTH MAPPINGS ARE NOT AUTOMATICALLY UPDATED	Medium	SOLVED - 11/09/2022
LACK OF FUNCTION PARAMETERS VALIDATION	Low	SOLVED - 11/09/2022
USE OF INSECURE FUNCTION	Low	SOLVED - 11/09/2022
REFLECTED UNSANITIZED INPUT	Low	SOLVED - 11/09/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) PYTH MAPPINGS ARE NOT AUTOMATICALLY UPDATED - MEDIUM

Description:

During the assessment, Halborn discovered that the Pyth Mappings in the airdropper project are not automatically updated each hour after the initial update on the application start. This is due to an error in the code which calculates whether an hour has passed since the last update.

The code subtracts the current time from the last update time, which always gives a negative result, and checks if this result is bigger than one hour.

This results in the airdropper not having the latest prices on the products it supports.

Code Location:

Listing 1: proxy-model.py/proxy/airdropper/airdropper.py

```
114     def try_update_pyth_mapping(self):
115         current_time = self.get_current_time()
116         if self.last_update_pyth_mapping is None or self.
 ↳ last_update_pyth_mapping - current_time > self.
 ↳ max_update_pyth_mapping_int:
117             try:
118                 self.pyth_client.update_mapping(self.
 ↳ pyth_mapping_account)
119                 self.last_update_pyth_mapping = current_time
```

Risk Level:

Likelihood - 2
Impact - 5

CVSS Vector:

- AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N

Recommendation:

The code should be updated so that the check on the time passed after the last Pyth Mapping update is calculated correctly:

Listing 2: proxy-model.py/proxy/airdropper/airdropper.py

```
116
117         if self.last_update_pyth_mapping is None or current_time -
 ↳  self.last_update_pyth_mapping > self.max_update_pyth_mapping_int:
```

Remediation Plan:

SOLVED The Neonlabs team solved this issue by making use of the absolute value of the time difference to prevent negative values.

Commit ID: 4e53565bbb56a7b9ea5f3d55fc4810cdc0863be3

3.2 (HAL-02) LACK OF FUNCTION PARAMETERS VALIDATION - LOW

During static analysis, it was identified that raw SQL queries were used, which could lead to a SQL Injection vulnerability. Halborn was unable to run dynamic tests against this, but it is recommended to remediate this issue and follow security best practices to increase the overall security posture of the application.

SQL injection (SQLi) refers to an injection attack in which an attacker can execute malicious SQL statements that control the database server of a web application and is implementation-dependent. The attacker, depending on their access rights, can read, delete, update database data/tables, or execute malicious commands.

The `is_airdrop_ready` function in the `/proxy/airdropper/airdropper.py` accepts the `eth_address` without any sanitization in the SQL statement, which could lead to a possible SQL injection attack.

In addition, it was identified that there was no sanitization in the `eth_address` variable/parameter. Halborn was unable to run dynamic tests against this, but it is recommended to remediate this issue and follow the security best practices to increase the overall security posture of the application.

No input validation can lead to an exception in the application and could also lead to injection attacks.

It was not possible to create both proofs of concept, but it is recommended to remediate this issue and follow security best practices to increase the overall security posture of the application and validate the user inputs against the requests.

Code Location:

Listing 3: /proxy/airdropper/airdropper.py (Line 58)

```
56     def is_airdrop_ready(self, eth_address):
57         with self._conn.cursor() as cur:
58             cur.execute(f"SELECT 1 FROM {self._table_name} WHERE
↳ eth_address = '{eth_address}'")
59             return cur.fetchone() is not None
```

Listing 4: /proxy/airdropper/airdropper.py (Line 58)

```
56     def is_airdrop_ready(self, eth_address):
57         with self._conn.cursor() as cur:
58             cur.execute(f"SELECT 1 FROM {self._table_name} WHERE
↳ eth_address = '{eth_address}'")
59             return cur.fetchone() is not None
```

Listing 5: /proxy/airdropper/airdropper.py (Line 37)

```
34     def airdrop_failed(self, eth_address, reason):
35         with self._conn.cursor() as cur:
36             cur.execute(f'''
37             INSERT INTO {self._table_name} (attempt_time,
↳ eth_address, reason)
38             VALUES ({datetime.now().timestamp()}, '{eth_address}',
↳  '{reason}')
39             ''')
```

Listing 6: /proxy/airdropper/airdropper.py (Line 52)

```
47     def register_airdrop(self, eth_address: str, airdrop_info:
↳ dict):
48         finished = int(datetime.now().timestamp())
49         duration = finished - airdrop_info['scheduled']
50         with self._conn.cursor() as cur:
51             cur.execute(f'''
52             INSERT INTO {self._table_name} (eth_address,
↳ scheduled_ts, finished_ts, duration, amount_galans)
53             VALUES ('{eth_address}', {airdrop_info['scheduled']},
↳ {finished}, {duration}, {airdrop_info['amount']})
54             ''')
```

CVSS Vector:

- AV:N/AC:H/PR:L/UI:N/S:U/C:L/I:L/A:N

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

It is recommended to use parameterized queries rather than raw queries when dealing with SQL queries. Parameterized queries allow the database to understand which parts of the SQL query should be considered as user input, thus solving SQL injection.
Moreover, it is recommended to add isAddress() function checks to check for ethereum addresses only and implement proper sanitization to prevent exceptions in the application.

Listing 7

```
1 ethers.utils.isAddress('0x8ba1f109551bd432803012645ac136ddd64dba72
↳ '); // true
2 web3.utils.isAddress('blah'); // false
```

In addition, it is recommended to add isAddress() function checks to check only ethereum addresses and implement proper sanitization.

Listing 8

```
1 ethers.utils.isAddress('0x8ba1f109551bd432803012645ac136ddd64dba72
↳ '); // true
2 web3.utils.isAddress('blah'); // false
```

Reference:

SQL Injection

parameterize-database-queries
web3-utils-isAddress
ethereum-address

Remediation Plan:

SOLVED The Neonlabs team solved this issue by implementing parameterized queries.

Commit ID: c8010ec579ef33dcc7a7214b19a98d0df0f947ef

3.3 (HAL-03) USE OF INSECURE FUNCTION - LOW

Description:

During the static analysis of the Proxy application code, it has been identified that the use of an insecure function could lead to command injection-type attacks. Halborn was unable to run dynamic tests against this, but it is recommended to remediate this issue and follow security best practices to increase the overall security posture of the application.

Python has many mechanisms to invoke an external executable. However, doing so can present a security issue if proper care is not taken to sanitize any user-provided or variable input.

This type of sub-process invocation is dangerous as it is vulnerable to various shell injection attacks. Great care should be taken to sanitize all inputs in order to mitigate this risk. Calls of this type are identified using certain commands which are known to use shells.

Code Location:

/proxy/common_neon/permission_token.py:49

Listing 9: /proxy/common_neon/permission_token.py (Line 49)

```
45     def mint_to(self, amount: int, ether_addr: Union[str,
↳ EthereumAddress], mint_authority_file: str, signer: SolanaAccount)
↳ :
46         token_account = self.create_account_if_needed(ether_addr,
↳ signer)
47         mint_command = f'spl-token mint "{str(self.token_mint)}" {
↳ Decimal(amount) * pow(Decimal(10), -9)}'
48         mint_command += f' --owner {mint_authority_file} -- "{str(
↳ token_account)}"'
49         os.system(mint_command)
```

CVSS Vector:

- AV:N/AC:H/PR:L/UI:N/S:U/C:L/I:L/A:N

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to follow the security best practices and add checks to sanitized and validate the input and additionally use the sub-process module rather than `os.system()` against inputs and don't use the shell, i.e `shell=False` parameter in the subprocess.Popen constructor.

Reference:

Python Subprocess
Use Subprocess securely

Remediation Plan:

SOLVED The Neonlabs team solved this issue by removing the os.system() function from the PermissionToken source code, as it was outdated and not used in the project repository.

Commit ID: 9bdca5bb215911ecb995c71f45ce6983dfc80057

3.4 (HAL-04) REFLECTED UNSANITIZED INPUT - LOW

Description:

We have detected that a number of the Neon labs Proxy web application handlers return user input to the web browser without sanitizing it.

Controllers accept HTTP POST requests containing user input, and their reply contains the same user input without sanitizing it first.

It is worth noting that we were unable to display the server's Response in the web browser (partly because the request is an **HTTP POST** request and because of the **Content-Type: application/json** response header). This, however, does not mean that the response is not displayed, or will not be displayed in the future, on a web page, making it a potential Cross Site Scripting attack vector.

Code Location:

When issuing a request with a malicious "id" value to the /solana endpoint, the server will reply with the malicious parameter unescaped to the web browser:

Listing 10: http://proxy:9090/solana (Line 24)

```
1  POST /solana HTTP/1.1
2  Host: 127.0.0.1:9090
3  User-Agent: curl/7.81.0
4  Accept: */*
5  Content-Length: 96
6  Content-Type: application/x-www-form-urlencoded
7  Connection: close
8
9  {
10  "id":"<script>alert(\"XSS\")</script>",
11  "jsonrpc":2.0,
12  "method":"web3_clientVersion",
13  "params":[]
```

```
14 }
15
16
17 HTTP/1.1 200
18 Content-Type: application/json
19 Access-Control-Allow-Origin: *
20 Content-Length: 126
21
22 {
23  "jsonrpc": "2.0",
24  "id": "<script>alert(\"XSS\")</script>",
25  "result": "Neon/v0.11.0-bf4f089ef06a8be4208e59c4a7f47a8efad245ce"
26 }
```

When issuing a request with a malicious “method” value to the /solana endpoint, the server will reply with an error with the malicious parameter unescaped to the web browser:

Listing 11: http://proxy:9090/solana (Line 26)

```
1 POST /solana HTTP/1.1
2 Host: 127.0.0.1:9090
3 User-Agent: curl/7.81.0
4 Accept: */*
5 Content-Length: 77
6 Content-Type: application/x-www-form-urlencoded
7 Connection: close
8
9 {
10  "id":1,
11  "jsonrpc":2.0,
12  "method":"<script>alert(\"XSS\")</script>",
13  "params":[]
14 }
15
16
17 HTTP/1.1 200
18 Content-Type: application/json
19 Access-Control-Allow-Origin: *
20 Content-Length: 124
21
22 {"jsonrpc": "2.0",
23 "id": 1,
```

```
24 "error": {
25  "code": -32601,
26  "message": "method <script>alert(\"XSS\")</script> is not
↳ supported"
27  }
28 }
```

Risk Level:

Likelihood - 1
Impact - 3

CVSS Vector:

- AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:N/A:N

Recommendation:

It is recommended to sanitize all user inputs which are reflected into the web browser page.
A good way to do this is to ensure that all such inputs are HTML-entity encoded before being reflected in a server response, but it is also recommended to utilize any front-end library functionality if available to mitigate this issue.

Reference:

Cross Site Scripting Prevention Cheat Sheet

Remediation Plan:

SOLVED The Neonlabs team solved this issue by adding sanitization of the RPC income values before working with them.

Commit ID: f7ae12e80b6d79b58c5453f45e13efa88ce24a06

THANK YOU FOR CHOOSING

// HALBORN