



HbarSuite

Backend Penetration Test

Prepared by: Halborn

Date of Engagement: August 18th, 2022 - September 12th, 2022

Visit: Halborn.com

DOCUMENT REVISION HISTORY	6
CONTACTS	6
1 EXECUTIVE OVERVIEW	8
1.1 INTRODUCTION	9
1.2 AUDIT SUMMARY	9
1.3 TEST APPROACH & METHODOLOGY	10
RISK METHODOLOGY	10
1.4 SCOPE	12
Smart Nodes	12
DEX / Oracle	12
Smart Rebalance Market Maker (SRMM)	12
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW	13
3 FINDINGS & TECH DETAILS	14
3.1 (HAL-01) DUPLICATED POOLS CREATED - HIGH	16
Description	16
Steps to reproduce	16
Screenshot	17
Risk Level	17
Recommendation	17
Remediation Plan	18
3.2 (HAL-02) POOLS BECOME UNAVAILABLE - HIGH	19
Description	19
Steps to reproduce	19
Screenshot	19

	Risk Level	20
	Recommendation	20
	Remediation Plan	20
3.3	(HAL-03) LARGE VALUES IN INPUT FIELDS SITE CRASH - MEDIUM	21
	Description	21
	Screenshot	21
	Risk Level	22
	Recommendation	22
	Remediation Plan	22
3.4	(HAL-04) UNAUTHENTICATED CACHE PURGING - MEDIUM	23
	Description	23
	Screenshot	23
	Risk Level	23
	Recommendation	23
	Remediation Plan	23
3.5	(HAL-05) LAUNCHPAD PURCHASE PAIRING SWAPPED - MEDIUM	24
	Description	24
	Screenshot	24
	Risk Level	25
	Recommendation	25
	Remediation Plan	26
3.6	(HAL-06) USDC DECIMALS - MEDIUM	27
	Description	27
	Screenshot	27
	Risk Level	27
	Recommendation	27

Remediation Plan	28
3.7 (HAL-07) NEGATIVE AMOUNTS IN INPUT FIELDS - LOW	29
Description	29
Screenshots	29
Risk Level	30
Recommendation	30
Remediation Plan	30
3.8 (HAL-08) NOT-A-NUMBER VALUES - LOW	31
Description	31
Screenshots	31
Risk Level	32
Recommendation	32
Remediation Plan	33
3.9 (HAL-09) MISSING CLICKJACKING PREVENTION - LOW	34
Description	34
Risk Level	34
Recommendation	34
Remediation Plan	35
3.10 (HAL-10) MISSING HTTP SECURITY HEADERS - LOW	36
Description	36
Risk Level	36
Recommendation	36
Remediation Plan	37
3.11 (HAL-11) CACHEABLE HTTPS RESPONSE - LOW	38
Description	38
Risk Level	38

	Recommendation	38
	Remediation Plan	38
3.12	(HAL-12) STRICT TRANSPORT SECURITY NOT ENFORCED - LOW	39
	Description	39
	Risk Level	39
	Recommendation	39
	Remediation Plan	39
3.13	(HAL-13) DEVELOPER LOGS - INFORMATIONAL	40
	Description	40
	Screenshot	40
	Risk Level	40
	Recommendation	40
	Remediation Plan	40
3.14	(HAL-14) OPEN PORTS - INFORMATIONAL	41
	Description	41
	Risk Level	43
	Recommendation	43
	Remediation Plan	43
3.15	(HAL-15) HISTORICAL IP INFORMATION - INFORMATIONAL	44
	Description	44
	Screenshot	44
	Risk Level	44
	Recommendation	45
	Remediation Plan	45
4	MANUAL TESTING	46
4.1	DENIAL OF SERVICE	47

4.2	WEB APPLICATION VULNERABILITIES	47
4.3	ACCESS CONTROL LEVELS / LOGIC FLAWS	47
4.4	SOURCE CODE REVIEW	48
4.5	DEX / ORACLE / SRMM	48
4.6	NFT	48
4.7	LIQUIDITY POOLS	49
4.8	SDK INTEGRATION	50

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	07/20/2022	Chris Meistre
0.2	Document Review	09/12/2022	Gokberk Gulgun
0.3	Draft Review	09/13/2022	Gabi Urrutia
1.0	Remediation Plan	09/15/2022	Chris Meistre
1.1	Remediation Plan Review	09/15/2022	Gokberk Gulgun
1.2	Remediation Plan Review	09/15/2022	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
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com



EXECUTIVE OVERVIEW



1.1 INTRODUCTION

HbarSuite engaged Halborn to conduct a security audit on the DEX and smart nodes, beginning on August 18th, 2022 and ending on September 12th, 2022 .

1.2 AUDIT SUMMARY

The team at Halborn assigned two full-time security engineers to audit the security of the HbarSuite DEX and smart nodes. These security engineers are blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols and web application security and architecture.

The goals of our security audits are to improve the quality of systems and aim for sufficient remediation to help protect users.

In summary, Halborn identified some security risks that were mostly addressed by the HbarSuite team.

The HbarSuite team were kept informed of both HIGH rated findings, and solved both issues.

While some automated testing was being performed, two of the smart nodes became unresponsive. The HbarSuite team were informed of this, but after their investigation it was ruled out that it could have been from the testing.

It was also found that there appears to be no timeouts being enforced on transactions that are waiting for manual user approval through the wallet software. A possible improvement on this could be to enforce a timeout based on the blocktime.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing 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 assist in quickly identifying potential issues. Halborn was also provided with an SDK to interact with the smart nodes. The following categories of vulnerabilities were evaluated during the audit:

- Mapping Application Content and Functionality.
- Application Logic Flaws.
- Access Handling.
- Authentication/Authorization Flaws.
- Rate Limitation Tests.
- Brute Force Attempts.
- Denial of Service and Resource Exhaustion Attempts.
- Safe Input Handling.
- Fuzzing of all input parameters.
- Injection (SQL/JSON/HTML/Command).
- Attack surface and publicly available services.
- Testing that pool ratio remain balanced.
- Testing for zero slippage.

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

Smart Nodes:

```
testnet-sn1.hbarsuite.network  
testnet-sn2.hbarsuite.network  
testnet-sn3.hbarsuite.network  
testnet-sn4.hbarsuite.network
```

DEX / Oracle:

<https://hsuite-finance.firebaseio.com>

- Exchange.
- Launchpad.

Smart Rebalance Market Maker (SRMM):

- Smart Rebalance Market Maker is a unique AMM algorithm, created in order to utilize native Atomic Swaps on Hedera.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	2	4	6	3

LIKELIHOOD

IMPACT

		(HAL-03) (HAL-04) (HAL-05) (HAL-06)		(HAL-01) (HAL-02)
(HAL-13) (HAL-14) (HAL-15)		(HAL-07) (HAL-08) (HAL-09) (HAL-10) (HAL-11) (HAL-12)		

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - DUPLICATED POOLS CREATED	High	SOLVED - 09/10/2022
HAL-02 - POOLS BECOME UNAVAILABLE	High	SOLVED - 09/11/2022
HAL-03 - LARGE VALUES IN INPUT FIELDS SITE CRASH	Medium	SOLVED - 09/15/2022
HAL-04 - UNAUTHENTICATED CACHE PURGING	Medium	SOLVED - 09/15/2022
HAL-05 - LAUNCHPAD PURCHASE PAIRING SWAPPED	Medium	SOLVED - 09/15/2022
HAL-06 - USDC DECIMALS	Medium	SOLVED - 09/15/2022
HAL-07 - NEGATIVE AMOUNTS IN INPUT FIELDS	Low	SOLVED - 09/15/2022
HAL-08 - NOT-A-NUMBER VALUES	Low	SOLVED - 09/15/2022
HAL-09 - MISSING CLICKJACKING PREVENTION	Low	SOLVED - 09/15/2022
HAL-10 - MISSING HTTP SECURITY HEADERS	Low	SOLVED - 09/15/2022
HAL-11 - CACHEABLE HTTPS RESPONSE	Low	SOLVED - 09/15/2022
HAL-12 - STRICT TRANSPORT SECURITY NOT ENFORCED	Low	SOLVED - 09/15/2022
HAL-13 - DEVELOPER LOGS	Informational	ACKNOWLEDGED
HAL-14 - OPEN PORTS	Informational	ACKNOWLEDGED
HAL-15 - HISTORICAL IP INFORMATION	Informational	ACKNOWLEDGED



FINDINGS & TECH DETAILS



3.1 (HAL-01) DUPLICATED POOLS CREATED - HIGH

Description:

It was found that by sending a valid `pool create` payload to the `/pools/create` API endpoint on all the smart nodes at the same, duplicated pools could be created.

This resulted in no positions being able to be opened in those duplicated pools.

Steps to reproduce:

A script was created to send a payload to the `/pools/create` API endpoint of `testnet-sn1`, `testnet-sn2`, `testnet-sn3` and `testnet-sn4` at virtually the same time.

This resulted in the pool being created through at least more than one of the nodes because it was observed that duplicated pools were created.

An example of the payload:

Listing 1

```
1 {"baseToken":{"id":"0.0.34332104"},"swapToken":{"id":"HBAR"}}
```

Screenshot:

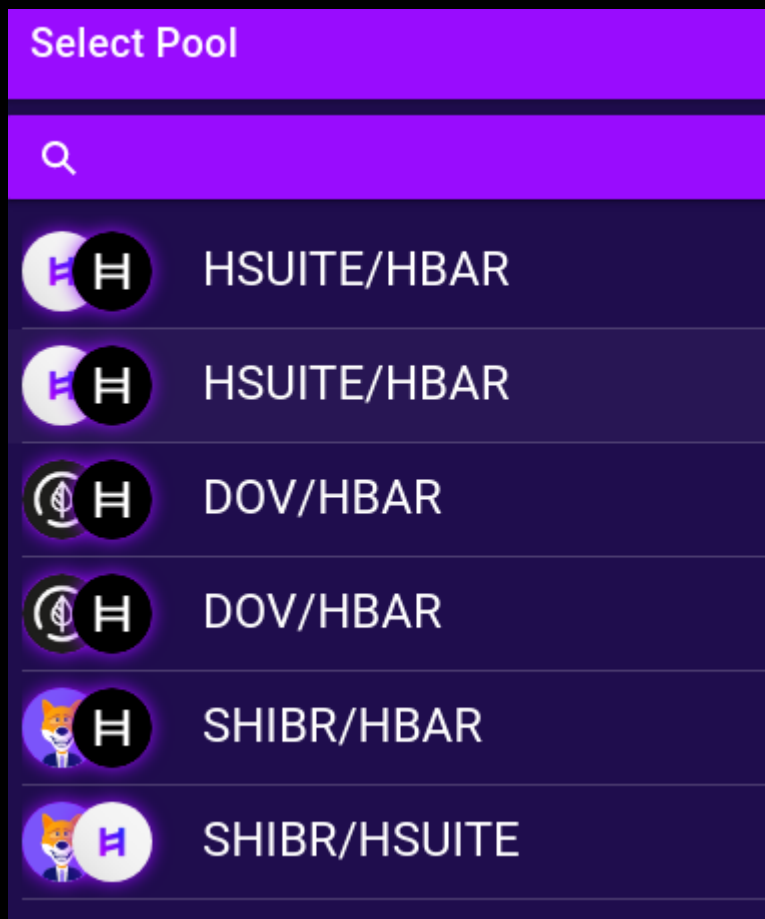


Figure 1: Duplicated pools showing on frontend

Risk Level:

Likelihood - 5

Impact - 3

Recommendation:

It is recommended to implement proper checks to make sure that duplicated pools are not created.

Remediation Plan:

SOLVED: While testing was ongoing, this issue was raised with the **HbarSuite team**. A fix was implemented and after retesting, it was noted that the same method could no longer be used to create a duplicated pool.

3.2 (HAL-02) POOLS BECOME UNAVAILABLE - HIGH

Description:

It was found, after some time, all the pools that were created while testing became unavailable. This varied from becoming unavailable any time between 3 hours after creation, up to 24 hours after creation.

This resulted in no positions being able to be opened in those duplicated pools.

Steps to reproduce:

A new pool was created. Monitoring was enabled using a custom-made script to check for available pools at an interval of 20 minutes.

During the test, it was observed that the created pool become unavailable after approximately 4 hours.

Screenshot:

```
1662824694.22724
[{"balance":{"tokens":[{"id":"0.0.34332104","balance":"10010100000","decimals":"4"}],"hbars":{"valueInTinybar":"400395992040"}},{"walletId":"0.0.48020045","feesId":"0.0.48020050","type":"FUNGIBLE_COMMON","name":"HSUITE/HBAR","environment":"testnet","created_at":"1661526005.000000000","is_running":true,"asset":{"pair":{"baseToken":{"id":"0.0.34332104","symbol":"HSUITE","ratio":"0.5","value":"0.003999996","amount":"1001010"},"swapToken":{"id":"HBAR","symbol":"HBAR","ratio":"0.5","value":"1.00001","amount":"4003.9599204"}},{"value":"8007.999919599204"}},{"balance":{"tokens":[{"id":"0.0.34719641","balance":"0","decimals":"4"}],"hbars":{"valueInTinybar":"0"}},{"walletId":"0.0.48216607","feesId":"0.0.48216613","type":"FUNGIBLE_COMMON","name":"USDC/HBAR","environment":"testnet","created_at":"1662807966.000000000","is_running":false,"asset":{"pair":{"baseToken":{"id":"0.0.34719641","symbol":"USDC","ratio":"0","value":"15.690212445476512","amount":"0"},"swapToken":{"id":"HBAR","symbol":"HBAR","ratio":"0","value":"1","amount":"0"}},{"value":"0"}}]}]
-----
1662825895.1984508
[]
-----
1662827096.1230915
[]
-----
1662828297.0102584
[]
-----
1662829497.9278226
[]
```

Figure 2: Output from the monitoring script showing pools become unavailable

Risk Level:**Likelihood - 5****Impact - 3****Recommendation:**

It is recommended to determine why the pools become unavailable as well as implement monitoring should the issue continue.

Remediation Plan:

SOLVED: While testing was ongoing, this issue was raised with the **HbarSuite team**. A fix was implemented and after retesting over a 24-hour period, it was noted that the pools remained available.

3.3 (HAL-03) LARGE VALUES IN INPUT FIELDS SITE CRASH – MEDIUM

Description:

It was found that <https://hsuite-finance.firebaseio.com> when large values are entered into the `amount` input fields, an error is generated and the site becomes unresponsive until a manual refresh of the browser. It appeared that the error was due to the number being converted into scientific notation.

Screenshot:

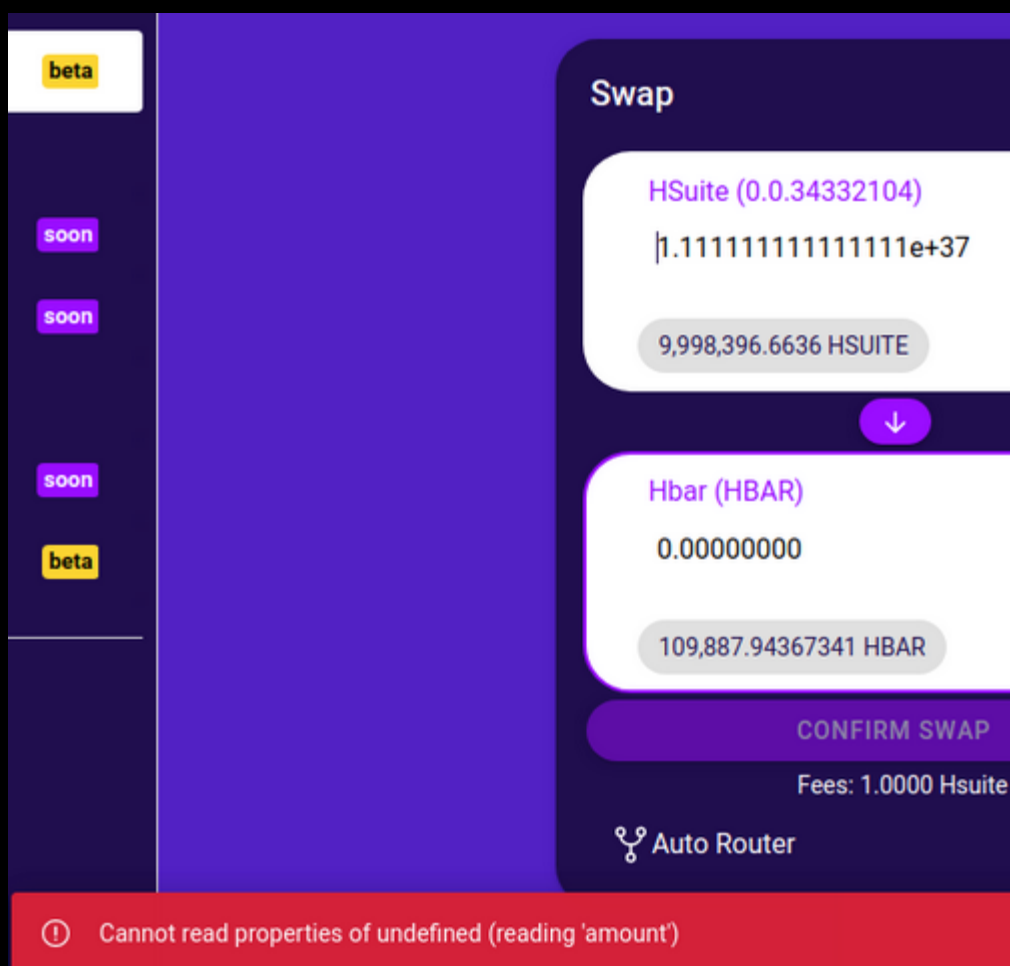


Figure 3: Big numbers causing an error

Risk Level:**Likelihood - 3****Impact - 3****Recommendation:**

It is recommended to ensure proper validation on all fields, and to implemented proper error handling so that a site does not become unresponsive due to errors.

Remediation Plan:

SOLVED: The **HbarSuite team** implemented a fix by using **encodingUrl** on all parameters.

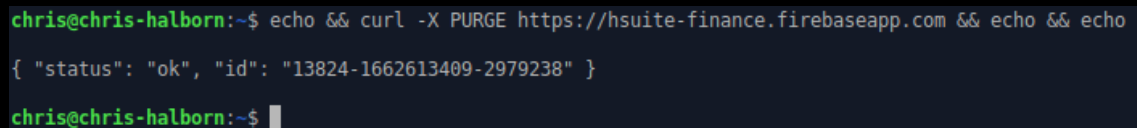
3.4 (HAL-04) UNAUTHENTICATED CACHE PURGING - MEDIUM

Description:

It was found that <https://hsuite-finance.firebaseio.com> allows unauthenticated cache purging. This allows an attacker to purge the site's caches.

This could lead to additional bandwidth costs and could also potential lead to Denial of Services.

Screenshot:



```
chris@chris-halborn:~$ echo && curl -X PURGE https://hsuite-finance.firebaseio.com && echo && echo
{ "status": "ok", "id": "13824-1662613409-2979238" }
chris@chris-halborn:~$
```

Figure 4: Unauthenticated cache purging

Risk Level:

Likelihood - 3

Impact - 3

Recommendation:

It is recommended to disable unauthenticated cache purging.

Remediation Plan:

SOLVED: The [HbarSuite team](#) implemented a fix by white listing only the official links on the smart nodes. By doing this, the Firebase application will no longer be operational. The official links are all under Cloudflare, where it is not possible for an anonymous user to purge the cache.

3.5 (HAL-05) LAUNCHPAD PURCHASE PAIRING SWAPPED – MEDIUM

Description:

It was found that when a negative value is provided when purchasing \$LPT in Launchpad on <https://hsuite-finance.firebaseio.com>, the tokens are swapped around and you will be spending \$LPT instead of earning it.

Screenshot:

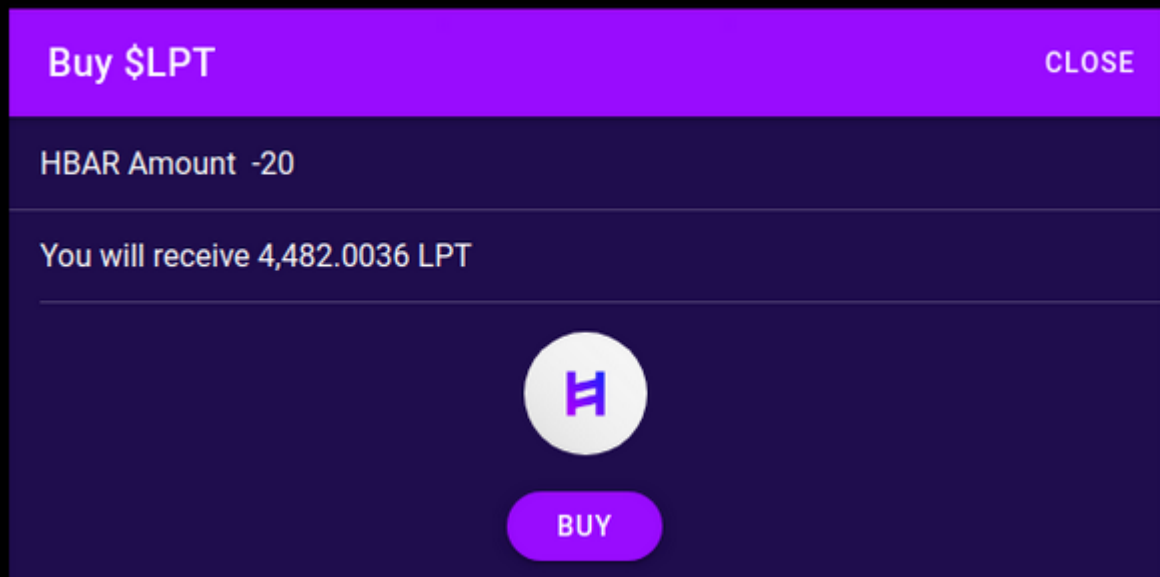


Figure 5: Enter a negative value to purchase

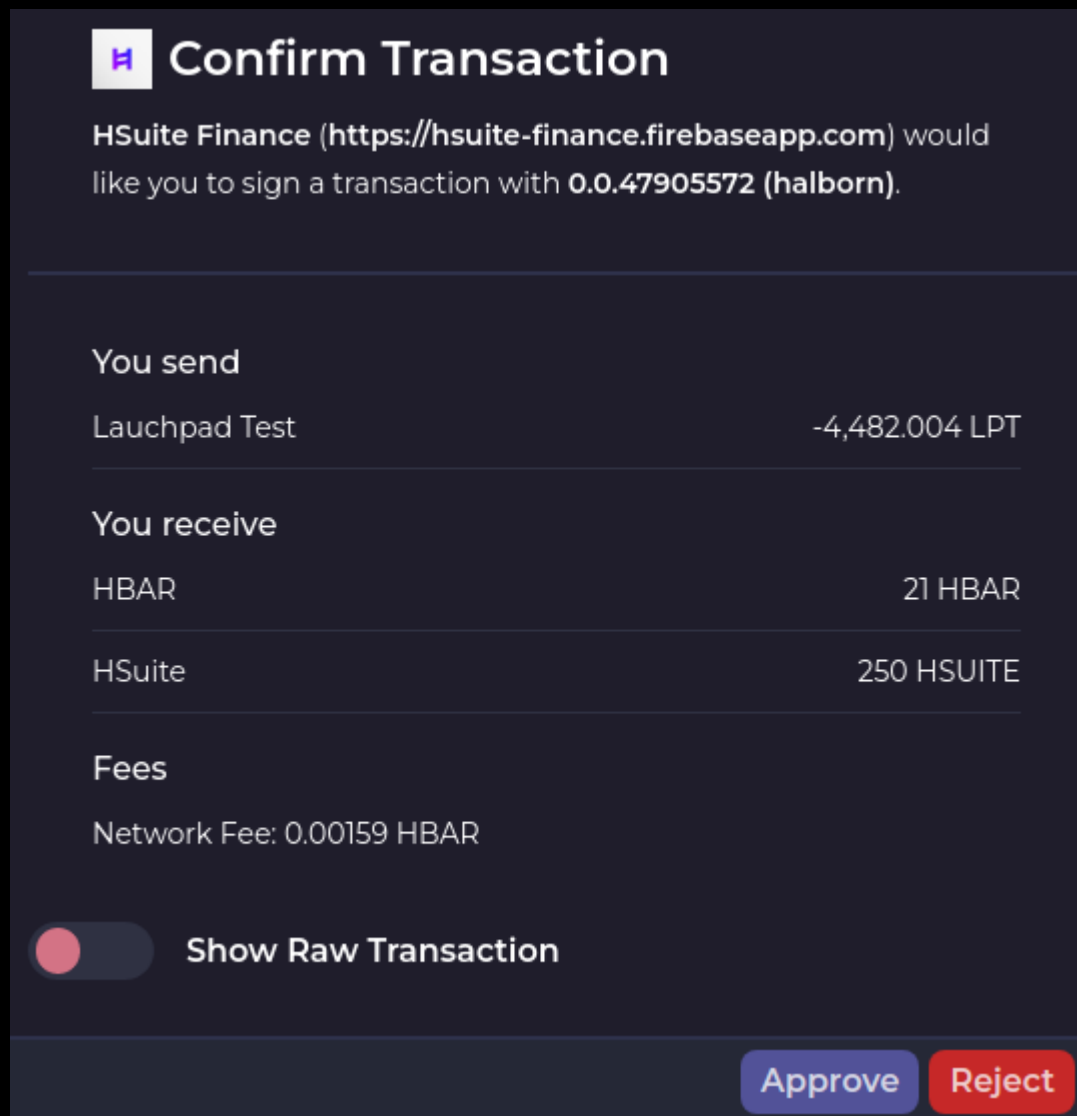


Figure 6: Transaction shows spending \$LPT

Risk Level:

Likelihood - 3

Impact - 3

Recommendation:

It is recommended to not allow a user to enter negative values for the amount inputs, as it might have unintended consequences for the user.

Remediation Plan:

SOLVED: The HbarSuite team implemented a fix by making sure no negative values can be entered into the input fields.

3.6 (HAL-06) USDC DECIMALS - MEDIUM

Description:

It was found that the USDC token has been set up with 4 decimals. This was observed in the `/tokens/list` API endpoint.

The accepted standard for the amount of decimals to be used when working with USDC is 6. By performing calculations using 4 decimal places, It might result in unexpected results.

Screenshot:

```
{
  "tokenId": "0.0.34719641",
  "image": "public/tokens/usdc.png",
  "name": "USDC",
  "symbol": "USDC",
  "website": "",
  "decimals": 4,
  "priceBotUrl": null,
  "price": 14.641202858103073,
  "coingecko_id": "usd-coin",
  "type": "FUNGIBLE_COMMON"
},
```

Figure 7: Enter a negative value to purchase

Risk Level:

Likelihood - 3

Impact - 3

Recommendation:

It is recommended to use 6 decimals when working with USDC.

Remediation Plan:

SOLVED: The HbarSuite team has fixed this by adjusting the decimal value of USDC to 6.

3.7 (HAL-07) NEGATIVE AMOUNTS IN INPUT FIELDS - LOW

Description:

It was found that <https://hsuite-finance.firebaseio.com> allowed for negative values to be entered in the `amount` input fields. When the transaction was performed, it did, however, use the positive representation of the value.

Screenshots:

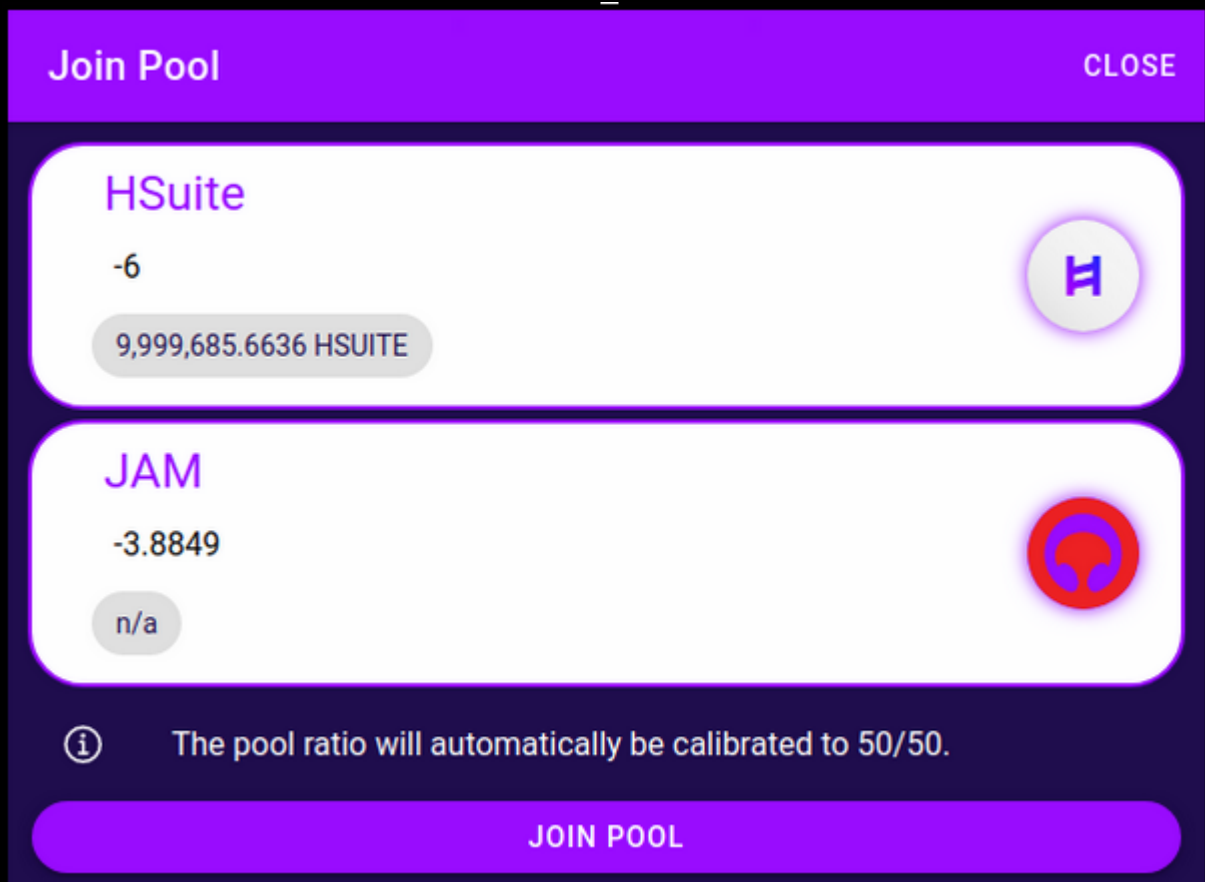


Figure 8: Join pool allows negative value

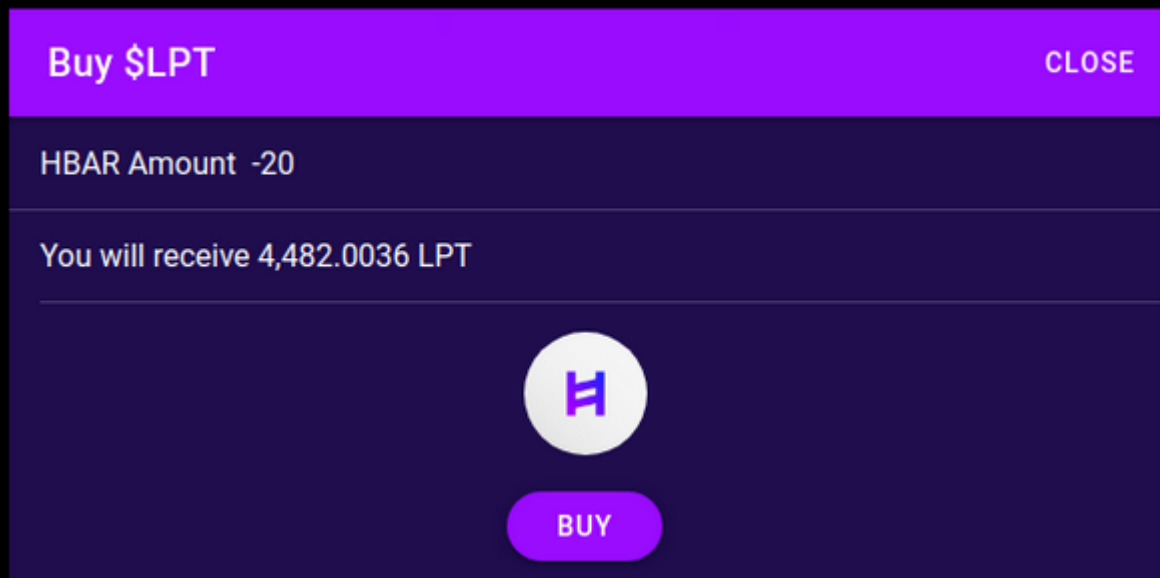


Figure 9: Launchpad allows negative value

Risk Level:

Likelihood - 3

Impact - 1

Recommendation:

It is recommended to not allow a user to enter negative values for the amount inputs, as it might have unintended consequences for the user.

Remediation Plan:

SOLVED: The [HbarSuite team](#) implemented a fix by making sure no negative values can be entered into the input fields.

3.8 (HAL-08) NOT-A-NUMBER VALUES - LOW

Description:

It was observed that when creating a new position on <https://hsuite-finance.firebaseio.com> the values being shown on the labels indicate incorrect values being parsed. They are showing **NaN** (Not-a-Number) values, instead of a properly formed numeric value.

Screenshots:

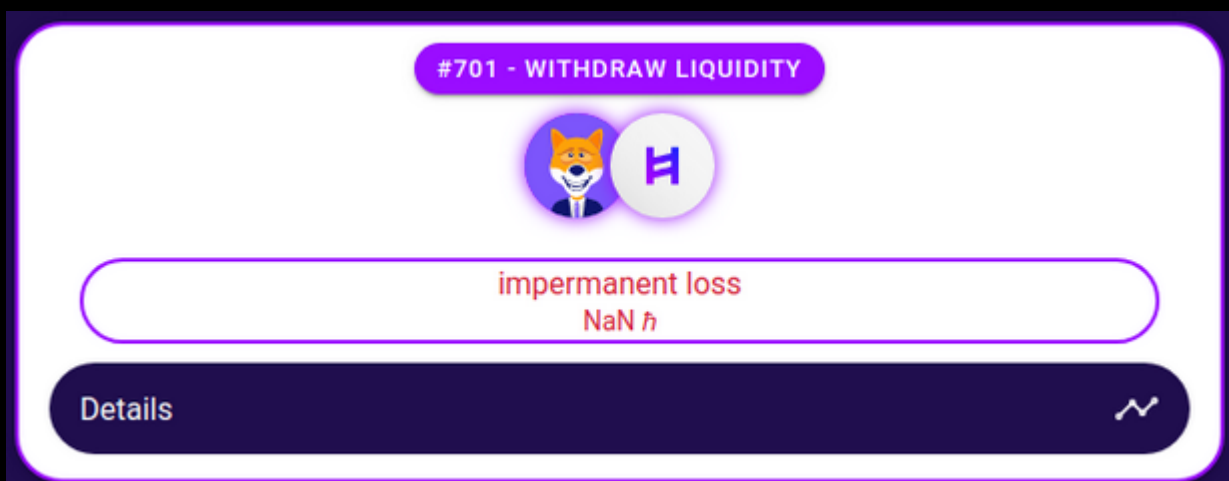


Figure 10: NaN values

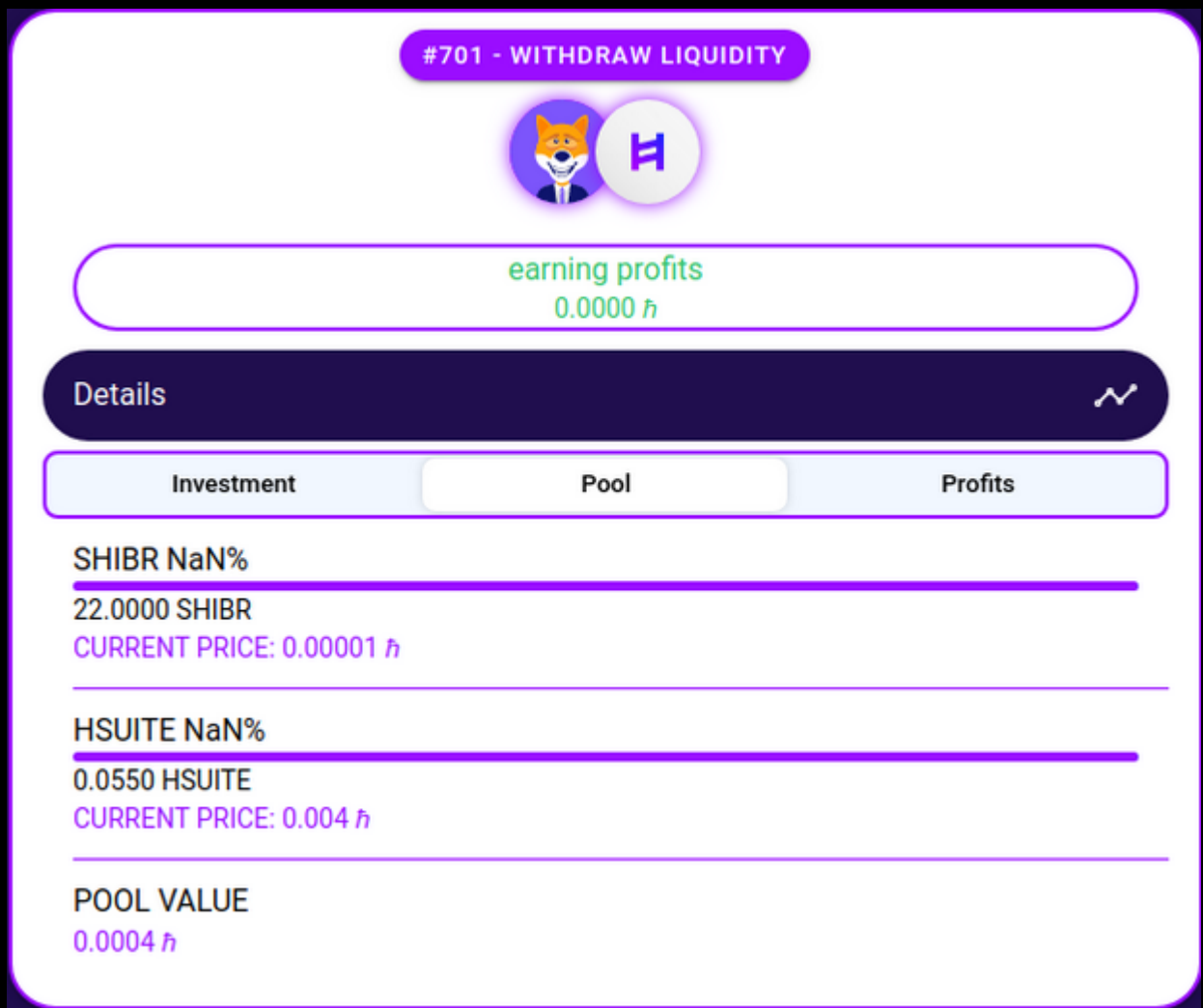


Figure 11: NaN values

Risk Level:**Likelihood - 3****Impact - 1****Recommendation:**

It is recommended that all values that are parsed are properly validated and formatted before displaying them to the user. It is also recommended to make sure that the error handling is done adequately.

Remediation Plan:

SOLVED: The **HbarSuite team** implemented a fix and no further **NaN** values were observed on the UI. There were, however, **NaN** errors being shown in the developer console of the browser, which the **HbarSuite team** has acknowledged.

3.9 (HAL-09) MISSING CLICKJACKING PREVENTION - LOW

Description:

It was found that <https://hsuite-finance.firebaseio.com> does not implement the anti-clickjacking **X-Frame-Options** header, which made it susceptible to clickjacking attacks.

Clickjacking is an attack that tricks a user into clicking a webpage element which is invisible or disguised as another element. This can cause users to unwittingly download malware, visit malicious web pages, provide credentials or sensitive information, transfer money, or purchase products online.

Typically, clickjacking is performed by displaying an invisible page or HTML element, inside an iframe, on top of the page the user sees. The user believes they are clicking the visible page, but in fact they are clicking an invisible element in the additional page transposed on top of it.

The invisible page could be a malicious page, or a legitimate page the user did not intend to visit - for example, a login page.

Risk Level:

Likelihood - 3

Impact - 1

Recommendation:

It is recommended to implement the **X-Frame-Options** header in all responses from the server, to protect users against clickjacking attacks.

Remediation Plan:

SOLVED: The HbarSuite team has fixed the issue by implementing the correct HTTP response headers.

3.10 (HAL-10) MISSING HTTP SECURITY HEADERS - LOW

Description:

It was found that <https://hsuite-finance.firebaseio.com> does not implement a number of the standard and recommended HTTP security headers.

The headers that were found to be missing are:

- access-control-expose-headers
- access-control-allow-methods
- access-control-allow-headers
- referrer-policy
- x-permitted-cross-domain-policies
- cross-origin-embedder-policy
- cross-origin-opener-policy
- access-control-allow-origin
- access-control-allow-credentials
- access-control-max-age
- permission-policy
- x-frame-options
- x-content-type-options
- cross-origin-resource-policy
- content-security-policy

Risk Level:

Likelihood - 3

Impact - 1

Recommendation:

It is recommended to implement the headers that are applicable to <https://hsuite-finance.firebaseio.com>, as this adds to the security and

reputation of the website.

Remediation Plan:

SOLVED: The HbarSuite team has fixed the issue by implementing the correct HTTP response headers.

3.11 (HAL-11) CACHEABLE HTTPS RESPONSE - LOW

Description:

It was found that responses from <https://hsuite-finance.firebaseio.com> may be cached by browsers. Unless directed otherwise, browsers may store a local cached copy of content received from web servers. Some browsers cache content accessed via HTTPS. If sensitive information in application responses is stored in the local cache, then this may be retrieved by other users who have access to the same computer at a future time.

Risk Level:

Likelihood - 3

Impact - 1

Recommendation:

Applications should return caching directives instructing browsers not to store local copies of any sensitive data. Often, this can be achieved by configuring the web server to prevent caching for relevant paths within the web root. Alternatively, most web development platforms allow you to control the server's caching directives from within individual scripts. Ideally, the web server should return the following HTTP headers in all responses containing sensitive content:

- Cache-control: no-store
- Pragma: no-cache

Remediation Plan:

SOLVED: The [HbarSuite team](#) has fixed the issue by implementing the correct HTTP response headers.

3.12 (HAL-12) STRICT TRANSPORT SECURITY NOT ENFORCED – LOW

Description:

The application running on `https://hsuite-finance.firebaseio.com` failed to prevent users from connecting to it over unencrypted connections. An attacker able to modify a legitimate user's network traffic could bypass the application's use of SSL/TLS encryption, and use the application as a platform for attacks against its users. This attack is performed by rewriting HTTPS links as HTTP, so that if a targeted user follows a link to the site from an HTTP page, their browser never attempts to use an encrypted connection.

Risk Level:

Likelihood - 3

Impact - 1

Recommendation:

The application should instruct web browsers to only access the application using HTTPS. To achieve this, enable HTTP Strict Transport Security (HSTS) by adding a response header with the name `Strict-Transport-Security` and the value `max-age=expireTime`, where `expireTime` is the time in seconds that browsers should remember that the site should only be accessed using HTTPS. Consider adding the `includeSubDomains` flag if appropriate.

Remediation Plan:

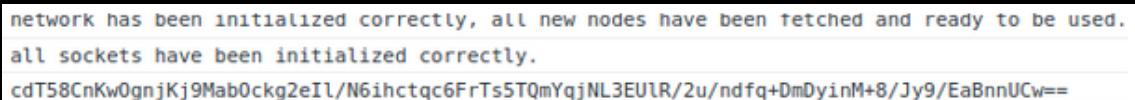
SOLVED: The `HbarSuite team` has fixed this by implementing the correct HTTP response headers.

3.13 (HAL-13) DEVELOPER LOGS - INFORMATIONAL

Description:

It was found on <https://hsuite-finance.firebaseio.com> it appears that some developer logs have been left enabled. This could lead to potentially sensitive information being made available to the user.

Screenshot:

A screenshot of a browser's developer console showing two log messages. The first message is "network has been initialized correctly, all new nodes have been fetched and ready to be used." and the second message is "all sockets have been initialized correctly." Both messages are in a light blue font. Below the messages is a long, alphanumeric string: "cdT58CnKw0gnjKj9Mab0ckg2eIl/N6ihctqc6FrTs5TQmYqjNL3EUlR/2u/ndfq+DmDyinM+8/Jy9/EaBnnUCw==".

```
network has been initialized correctly, all new nodes have been fetched and ready to be used.  
all sockets have been initialized correctly.  
cdT58CnKw0gnjKj9Mab0ckg2eIl/N6ihctqc6FrTs5TQmYqjNL3EUlR/2u/ndfq+DmDyinM+8/Jy9/EaBnnUCw==
```

Figure 12: Developer logs being shown in browser console

Risk Level:

Likelihood - 1

Impact - 1

Recommendation:

It is recommended to ensure that any information that is being displayed to a user is not anything sensitive, and that developer logs should be disabled in a production environment.

Remediation Plan:

ACKNOWLEDGED: The [HbarSuite Team](#) has acknowledged this finding.

3.14 (HAL-14) OPEN PORTS - INFORMATIONAL

Description:

The three provided hosts were scanned to see which ports are open and which services were running.

testnet-sn1.hbarsuite.network (104.21.73.174):

Port	Service Banner
80	cloudflare
443	cloudflare
2052	No banner
2053	nginx
2082	No banner
2083	nginx
2086	No banner
2087	nginx
2095	No banner
2096	nginx
8080	cloudflare
8443	cloudflare
8880	No banner

testnet-sn2.hbarsuite.network (104.21.73.174):

Port	Service Banner
80	cloudflare
443	cloudflare
2052	No banner
2053	nginx
2082	No banner
2083	nginx
2086	No banner
2087	nginx
2095	No banner
2096	nginx
8080	cloudflare
8443	cloudflare
8880	No banner

testnet-sn3.hbarsuite.network (104.21.73.174):

Port	Service Banner
80	cloudflare
443	cloudflare
2052	No banner
2053	nginx
2082	No banner
2083	nginx
2086	No banner
2087	nginx
2095	No banner
2096	nginx
8080	cloudflare
8443	cloudflare
8880	No banner

testnet-sn4.hbarsuite.network (172.67.146.161):

Port	Service Banner
80	cloudflare
443	cloudflare
2052	No banner
2053	nginx
2082	No banner
2083	nginx
2086	No banner
2087	nginx
2095	No banner
2096	nginx
8080	cloudflare
8443	cloudflare
8880	No banner

Risk Level:

Likelihood - 1

Impact - 1

Recommendation:

It is recommended that any services that provide administrative functions, should only be accessible through a VPN.

Remediation Plan:

ACKNOWLEDGED: The HbarSuite Team has acknowledged this finding.

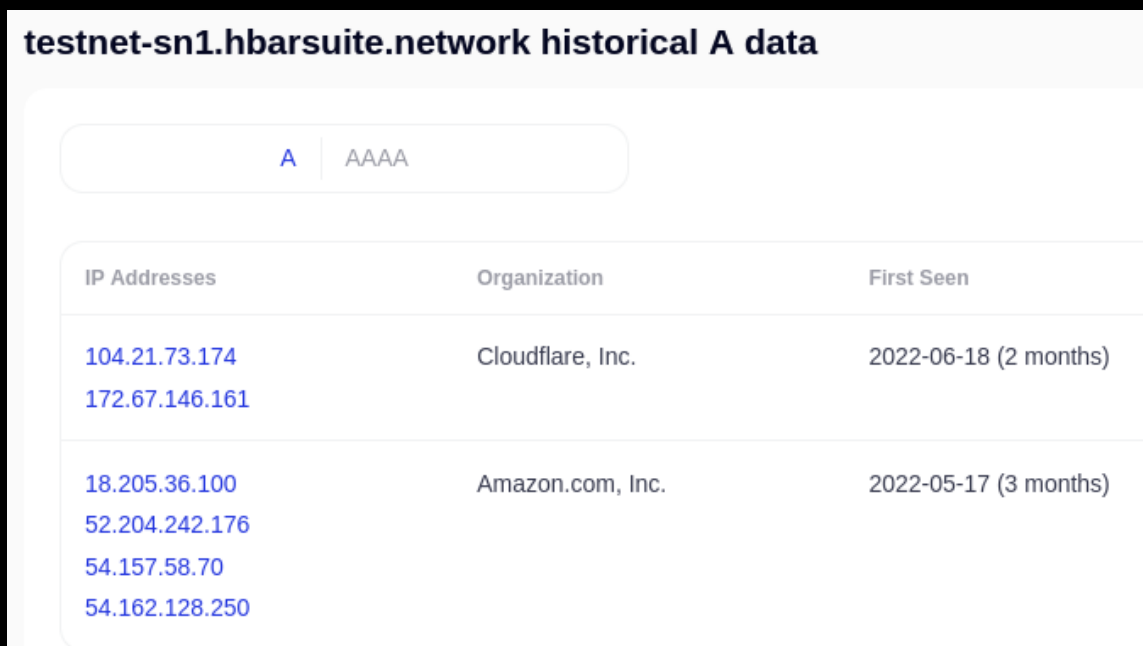
3.15 (HAL-15) HISTORICAL IP INFORMATION – INFORMATIONAL

Description:

It was found that previous IP addresses that have been associated with the smart nodes were accessible in publicly available DNS history records. Using this information, the direct IP addresses of the nodes were found.

It should, however, be noted that the Cloudflare proxy and firewall has been set up to correctly to not allow direct access to these nodes using those IP addresses.

Screenshot:



The screenshot displays a web interface titled "testnet-sn1.hbarsuite.network historical A data". It features a tabbed interface with "A" selected and "AAAA" as an alternative. Below the tabs is a table with three columns: "IP Addresses", "Organization", and "First Seen". The table lists two entries: one for Cloudflare, Inc. with IP addresses 104.21.73.174 and 172.67.146.161, first seen on 2022-06-18 (2 months ago); and another for Amazon.com, Inc. with IP addresses 18.205.36.100, 52.204.242.176, 54.157.58.70, and 54.162.128.250, first seen on 2022-05-17 (3 months ago).

IP Addresses	Organization	First Seen
104.21.73.174 172.67.146.161	Cloudflare, Inc.	2022-06-18 (2 months)
18.205.36.100 52.204.242.176 54.157.58.70 54.162.128.250	Amazon.com, Inc.	2022-05-17 (3 months)

Figure 13: Historical IP address information for testnet-sn1

Risk Level:

Likelihood - 1

Impact - 1

Recommendation:

It is recommended to ensure that direct access to the smart nodes via their IP addresses should remain blocked.

Remediation Plan:

ACKNOWLEDGED: The **HbarSuite Team** has acknowledged this finding.



MANUAL TESTING



4.1 DENIAL OF SERVICE

In order to test the nodes against vulnerabilities against denial of service attacks, a variety of tests were performed. This included testing to see that rate-limiting was being applied that Cloudflare WAF rules were set up correctly, and that direct access to the nodes were not allowed. It was also tested to see if API endpoints could be manipulated to achieve resource exhaustion on the underlying server.

4.2 WEB APPLICATION VULNERABILITIES

A variety of tests were performed to determine if the web application hosted at <https://hsuite-finance.firebaseio.com> is vulnerable to any of the well-known vulnerabilities. This included checking for injection vulnerabilities (SQL injection, XSS injection, prototype pollution, HTML injection and command injection), assuring safe input handling, missing security headers, sensitive information being stored in local storage and fuzzing all input parameters.

The **Smart Nodes** underwent the same tests to ensure that the API endpoints were not vulnerable to any of these attacks.

4.3 ACCESS CONTROL LEVELS / LOGIC FLAWS

Tests were conducted to ensure that there were no logic flaws. For example, making sure that one user cannot withdraw another user's earnings from a liquidity pool, an NFT cannot be created on behalf of another user, etc.

4.4 SOURCE CODE REVIEW

During the testing, the source code (Javascript) that was available through the web browser was analyzed for general and well-known bugs. This included trying to determine new undocumented API endpoints, flaws in encryption methods used, hardcoded credentials or secrets and proper validation of user input.

4.5 DEX / ORACLE / SRMM

The SRMM is a solution that was implemented to keep the pool at a balanced ratio of 50/50. This would also provide a zero-slippage effect. The test environment was set up with a 5% slippage allowance.

This was tested by a monitoring script that was set up to poll the ratio on the pools that were being tested. At the same time, swaps and trades were being done, to simulate the actions required for the ratios to become unbalanced. During the testing, it was observed that the ratios remained balanced.

The slippage was also being manually verified to ensure that there was effectively zero slippage.

4.6 NFT

A NFT is created for each position within a pool. The liquidity is associated with the NFT, and not the wallet, therefore allowing a user to transfer their liquidity to another.

It was tested to make sure the ownership of the NFTs are thoroughly checked, by trying to send the NFT outside the application, and at the same time try to make a withdrawal or close the position from the liquidity pool.

It was also established that once an NFT has been burned, it is not possible to still withdraw from and transfer the NFT.

The following tests cases are completed through UI/Desktop/SDK.

- Withdraw/Burn NFT multiple times.
- Burn NFT without withdraw.
- Close Position After NFT transfer.
- Concurrency testing with multiple operations.
- LP Position Ownership testing.
- Fake Amount of Withdrawals Tests from UI/SDK.

4.7 LIQUIDITY POOLS

It was tested to ensure that one cannot withdraw more from the pool than is allowed (i.e., invested and earnings) and that fake tokens cannot be introduced into the pools. It was tested to make sure that one user, withdrawing from the pool, does not affect another user's liquidity.

4.8 SDK INTEGRATION

An SDK, together with a sample application, was provided to assist in automating some of the testing.

An example of the SDK being used:

Listing 2

```
1 import { Component } from '@angular/core';
2 import { SmartNodeSdkService } from '@hsuite/angular-sdk';
3 import Decimal from 'decimal.js';
4 @Component({
5   selector: 'app-root',
6   templateUrl: 'app.component.html',
7   styleUrls: ['app.component.css']
8 })
9 export class AppComponent {
10   title = "title"
11   public accountIds: Array<string> = new Array();
12   public network: string = 'testnet';
13
14   constructor(
15     private smartNodeSdkService: SmartNodeSdkService,
16   ) {
17     const sender = "0.0.47905572"
18
19     // HSUITE/JAM
20     const poolWalletId = "0.0.48222538"
21     const dov = "0.0.34719665"
22     const usdc = "0.0.34719641"
23
24
25     // subscribing to webSockets authentication events...
26     this.smartNodeSdkService.getEventsObserver().subscribe(async(
27 ↪ event) => {
28       switch(event.method) {
29         case 'authentication':
30           console.log('authentication', event);
31           break;
32         case 'authenticate':
33           console.log('authenticate', event);
34           break;
```

```

34         case 'events':
35             console.log('events', event);
36             break;
37         case 'error':
38             console.log('error', event);
39             break;
40     }
41 });
42
43 // load pools info...
44 this.smartNodeSdkService.getRestService().loadPools().then(
↳ pools => {
45     console.log("pools: ", pools);
46 }).catch(error => console.error(error));
47
48 setTimeout(async () => {
49     // get nodes online...
50     console.log("nodes online");
51     console.log(this.smartNodeSdkService.getSocketsService().
↳ getNodesOnline());
52
53     // get current node...
54     console.log("current node");
55     console.log(this.smartNodeSdkService.getNetworkService().
↳ getCurrentNode());
56
57     // get Hashpack Service...
58     console.log("connecting");
59     await this.smartNodeSdkService.getHashPackService().connect(
60         <'previewnet' | 'testnet' | 'mainnet'> 'testnet',
61         'hashpack'
62     );
63
64 }, 5000);
65
66 setTimeout(async () => {
67     // nfts
68     console.log("nfts");
69     await this.smartNodeSdkService.getHederaService().
↳ getAccountNftTokens(sender).then(nfts => {
70         console.log(nfts);
71     });
72
73     console.log("burn nft 20");

```

```

74     await this.smartNodeSdkService.burnLpNft(20).then(output =>
75 ↪ {
76         console.log("burnLpNft 20", output);
77     });
78
79     console.log("burn nft 718");
80     await this.smartNodeSdkService.burnLpNft(718).then(output =>
81 ↪ {
82         console.log("burnLpNft", output);
83     });
84 }, 20000);
85
86 setTimeout(async () => {
87
88     const mintPool = {
89         baseToken: {
90             id: dov,
91             amount: new Decimal(99),
92             decimals: new Decimal(4)
93         },
94         swapToken: {
95             id: usdc,
96             amount: new Decimal(0.3765),
97             decimals: new Decimal(4)
98         }
99     }
100
101     console.log("mint nft", mintPool);
102     await this.smartNodeSdkService.mintLpNft(mintPool).then(pool
103 ↪ => {
104         console.log("mintLpNft", pool);
105
106         const nftPool = {
107             tokenId: pool.tokenId,
108             serialNumber: pool.serialNumber
109         }
110
111         // join pool
112         const joinPool = {
113             baseToken: {
114                 id: dov,
115                 amount: new Decimal(99),

```

```

115         decimals: 4
116     },
117     swapToken: {
118         id: usdc,
119         amount: new Decimal(0.3765),
120         decimals: 4
121     }
122 }
123
124     console.log("joining pool", joinPool, nftPool);
125     this.smartNodeSdkService.getHederaService().
126     ↪ joinPoolTransaction(sender, poolWalletId, joinPool, nftPool).then(
127     ↪ output => {
128         console.log("joining pool", output);
129     });
130
131     // nfts
132     console.log("nfts");
133     await this.smartNodeSdkService.getHederaService().
134     ↪ getAccountNftTokens(sender).then(nfts => {
135         console.log(nfts);
136     });
137 }, 15000);
138
139 */
140 }
141 }

```



THANK YOU FOR CHOOSING

// HALBORN

