



0x_Nodes

System11 Frontend Dapp Testing

Prepared by: Halborn

Date of Engagement: October 19th, 2021 - October 29th, 2021

Visit: Halborn.com

DOCUMENT REVISION HISTORY	3
CONTACTS	3
1 EXECUTIVE OVERVIEW	4
1.1 INTRODUCTION	5
1.2 AUDIT SUMMARY	5
1.3 TEST APPROACH & METHODOLOGY	6
RISK METHODOLOGY	6
1.4 SCOPE	8
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW	9
3 FINDINGS & TECH DETAILS	10
3.1 (HAL-01) HARDCODED API KEY FOUND - HIGH	11
Description	11
Code Location	11
Recommendation	11
3.2 (HAL-02) NO RATE LIMITING ON APIs - MEDIUM	12
Description	12
Results	12
Risk Level	13
Recommendations	13
3.3 (HAL-03) WEAK CIPHER SUITES ON TLS PROTOCOLS - INFORMATIONAL	14
Description	14
Attached evidence	14
Risk Level	16

Recommendations	16
3.4 (HAL-04) AUTOMATED TESTING RESULTS - INFORMATIONAL	17
Description	17
Risk Level	17
Results	17

DRAFT

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	10/20/2021	Nishit Majithia
0.2	Document Edits	10/21/2021	Nishit Majithia
0.3	Document Edits	10/25/2021	Nishit Majithia
0.4	Final Draft	10/29/2021	Nishit Majithia
0.5	Final Draft Review	10/29/2021	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
Nishit Majithia	Halborn	Nishit.Majithia@halborn.com



EXECUTIVE OVERVIEW

1.1 INTRODUCTION

0x_nodes engaged Halborn to conduct a security assessment on web frontend beginning on October 19th, 2021 and ending October 29th, 2021. 0x_nodes is a system that allows users to simplify access to defi yields. Users deposit assets to the system, the assets are put into various yield-generating strategies, and the users collect rewards.

The security assessment was scoped to the web frontend code provided in the Github repository [0xNodes Web Frontend](#)

1.2 AUDIT SUMMARY

The team at Halborn was provided five weeks for the engagement and assigned a full time security engineer to audit the security of the web application. 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 audit to achieve the following:

- Ensure that web application functions operate as intended.
- Identify potential security issues with the web application.

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverables set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure web application development.

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 penetration testing. 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:

- Mapping Application Content and Functionality
- Side/Browser Based Control Auditing
- Application Logic Flaws
- Access Handling
- Light Brute Force Attacks
- Input Handling
- Fuzzing of all input parameters
- Test for Injection (SQL/JSON/HTML/Command)
- Technology stack specific vulnerabilities and Code Audit
- 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. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics

that was 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 the git repository:

- <https://github.com/0xNODES/platform>
- commit ID: `6bc0727e0c42963c788cf47fe44f0cfef79e337d`

DRAFT

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	1	0	2

LIKELIHOOD

IMPACT

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

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HARDCODED API KEY FOUND	High	-
NO RATE LIMITING ON APIs	Medium	-
WEAK CIPHER SUITES ON TLS PROTOCOLS	Informational	-
AUTOMATED TESTING RESULTS	Informational	-



FINDINGS & TECH DETAILS

3.1 (HAL-01) HARDCODED API KEY FOUND - HIGH

Description:

The Infura API provides access to the Ethereum network over HTTPS and Web-Socket. The free version has limit of 100,000 requests/day while premium versions have from 200,000 until 5,000,000 request/day. If the Infura API keys are compromised, they can be used for other users causing a Denial of Service (DoS) denying administrators post-deployment administrative tasks.

Infura API keys found in `.env` environment file as `REACT_APP_INFURA_API_KEY` variable. This is used when the Fallback Provider is triggered. The Infura API key is mainly used to read data from the blockchain.

Code Location:

Listing 1: `.env`

```
13 REACT_APP_MODULE_MAP_ADDRESSES='{ "1": "0
    x037254eB3adEBF06Eda6ECE3029355888aEeFa70
    ", "3": "", "4": "", "42": "0
    x724D70dE0913E231e9323E71Ef399d426677f58E" }'
14
15 REACT_APP_INFURA_API_KEY="fdf88904416b4xxxxxxxxxx5ba3d6947"
16 REACT_APP_ALCHEMY_API_KEY=""
17 REACT_APP_ETHERSCAN_API_KEY=""
```

Recommendation:

Remove the hardcoded API key and make it available for user to fill this value runtime. If it is deployed in AWS then recommended to use AWS secret Manager to store secrets.

3.2 (HAL-02) NO RATE LIMITING ON APIs - MEDIUM

Description:

Exploiting the **lack of rate limiting** through APIs, it is possible to force the application to crash due to the collapse of the network, CPU, memory or storage resources. This often results in a sluggish behavior, system crashes, or other rogue server behaviors, resulting in a denial of service (DoS).

Results:

Results

Target

Positions

Payloads

Resource Pool

Options

Filter: Showing all items

?

Request ^	Payload	Status	Error	Timeout	Length	Comment
61	61	200	<input type="checkbox"/>	<input type="checkbox"/>	2115	
62	62	200	<input type="checkbox"/>	<input type="checkbox"/>	2115	
63	63	200	<input type="checkbox"/>	<input type="checkbox"/>	2115	
64	64	200	<input type="checkbox"/>	<input type="checkbox"/>	2115	
65	65	200	<input type="checkbox"/>	<input type="checkbox"/>	2115	
66	66	200	<input type="checkbox"/>	<input type="checkbox"/>	2115	
67	67	200	<input type="checkbox"/>	<input type="checkbox"/>	2115	
68	68	200	<input type="checkbox"/>	<input type="checkbox"/>	2115	
69	69	200	<input type="checkbox"/>	<input type="checkbox"/>	2115	
70	70	200	<input type="checkbox"/>	<input type="checkbox"/>	2115	
71	71	200	<input type="checkbox"/>	<input type="checkbox"/>	2115	
72	72	200	<input type="checkbox"/>	<input type="checkbox"/>	2115	
73	73	200	<input type="checkbox"/>	<input type="checkbox"/>	2115	
74	74	200	<input type="checkbox"/>	<input type="checkbox"/>	2115	
75	75	200	<input type="checkbox"/>	<input type="checkbox"/>	2115	
76	76	200	<input type="checkbox"/>	<input type="checkbox"/>	2115	
77	77	200	<input type="checkbox"/>	<input type="checkbox"/>	2115	
78	78	200	<input type="checkbox"/>	<input type="checkbox"/>	2115	
79	79	200	<input type="checkbox"/>	<input type="checkbox"/>	2115	

Request

Response

Pretty

Raw

Hex

\n

☰

1 GET / HTTP/1.1

2 Host: 192.168.47.128:3000

3 Upgrade-Insecure-Requests: 1

4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.61 Safari/537.368

5 Accept:

text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9

6 Accept-Encoding: gzip, deflate

?

⚙

⬅

➡

Search...

0 matches

The result shows one of the API, but on all API endpoints, it is applicable

Risk Level:

Likelihood - 4

Impact - 3

Recommendations:

Each API that is deployed must have its throttle rates defined in its code. This could include parameters such as execution timeouts, maximum memory allowed, the number of records per page that can be returned to a user, or the number of processes allowed within a defined timeframe.

Also, it is recommended to setup policies that would limit the massive use of APIs and include automated locks to prevent abuse.

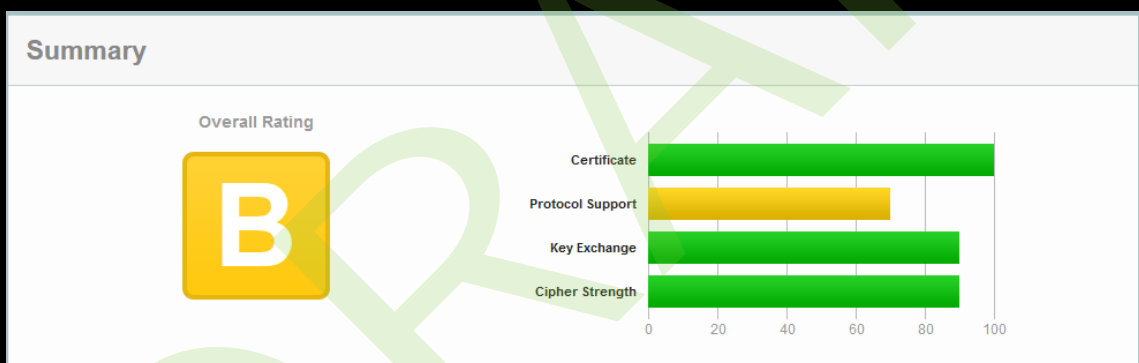
3.3 (HAL-03) WEAK CIPHER SUITES ON TLS PROTOCOLS - INFORMATIONAL

Description:

Cipher suites used for TLS protocols v1.0, v1.1 and v1.2 in [system11.0xnodes.io](#) and [www.0xnodes.io](#) are considered weak because they are vulnerable to some **time attacks**. However, the impact is relatively minimal because this vulnerability requires some very particular scenarios to be exploited.

Attached evidence:

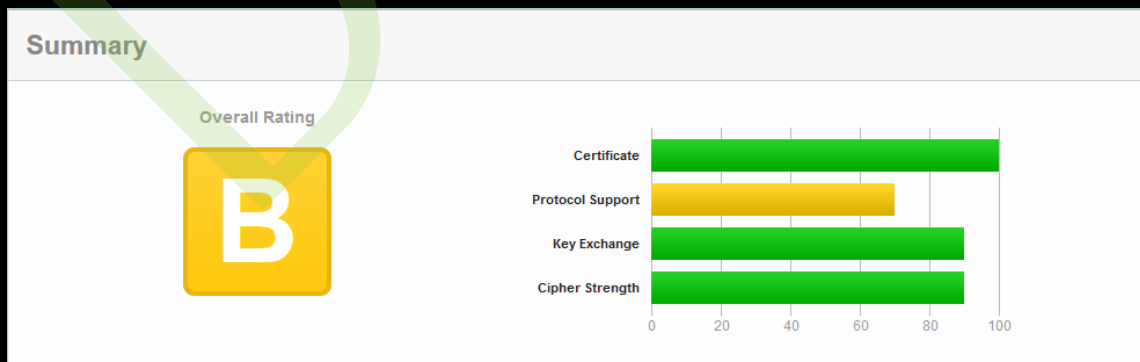
Weak cipher suite on [system11.0xnodes.io](#):




Cipher Suites			
# TLS 1.3 (server has no preference)			
TLS_AES_128_GCM_SHA256 (0x1301)	ECDH x25519 (eq. 3072 bits RSA)	FS	128
TLS_AES_256_GCM_SHA384 (0x1302)	ECDH x25519 (eq. 3072 bits RSA)	FS	256
TLS_CHACHA20_POLY1305_SHA256 (0x1303)	ECDH x25519 (eq. 3072 bits RSA)	FS	256
# TLS 1.2 (suites in server-preferred order)			
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)	ECDH x25519 (eq. 3072 bits RSA)	FS	128
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)	ECDH x25519 (eq. 3072 bits RSA)	FS	256 ^P
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)	ECDH x25519 (eq. 3072 bits RSA)	FS	256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)	ECDH x25519 (eq. 3072 bits RSA)	FS	128
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)	ECDH x25519 (eq. 3072 bits RSA)	FS	256
TLS_RSA_WITH_AES_128_GCM_SHA256 (0x9c)		WEAK	128
TLS_RSA_WITH_AES_256_GCM_SHA384 (0x9d)		WEAK	256
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)		WEAK	128
TLS_RSA_WITH_AES_256_CBC_SHA (0x35)		WEAK	256
TLS_RSA_WITH_3DES_EDE_CBC_SHA (0xa)		WEAK	112
# TLS 1.1 (suites in server-preferred order)			
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)	ECDH x25519 (eq. 3072 bits RSA)	FS	128
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)	ECDH x25519 (eq. 3072 bits RSA)	FS	256
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)		WEAK	128
TLS_RSA_WITH_AES_256_CBC_SHA (0x35)		WEAK	256
TLS_RSA_WITH_3DES_EDE_CBC_SHA (0xa)		WEAK	112
# TLS 1.0 (suites in server-preferred order)			
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)	ECDH x25519 (eq. 3072 bits RSA)	FS	128
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)	ECDH x25519 (eq. 3072 bits RSA)	FS	256
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)		WEAK	128
TLS_RSA_WITH_AES_256_CBC_SHA (0x35)		WEAK	256
TLS_RSA_WITH_3DES_EDE_CBC_SHA (0xa)		WEAK	112

(P) This server prefers ChaCha20 suites with clients that don't have AES-NI (e.g., Android devices)

Weak cipher suite on www.0xnodes.io:



 Cipher Suites	
# TLS 1.3 (server has no preference)	
TLS_AES_128_GCM_SHA256 (0x1301) ECDH x25519 (eq. 3072 bits RSA) FS	128
TLS_AES_256_GCM_SHA384 (0x1302) ECDH x25519 (eq. 3072 bits RSA) FS	256
TLS_CHACHA20_POLY1305_SHA256 (0x1303) ECDH x25519 (eq. 3072 bits RSA) FS	256
# TLS 1.2 (suites in server-preferred order)	
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f) ECDH x25519 (eq. 3072 bits RSA) FS	128
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8) ECDH x25519 (eq. 3072 bits RSA) FS	256 ^P
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030) ECDH x25519 (eq. 3072 bits RSA) FS	256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013) ECDH x25519 (eq. 3072 bits RSA) FS WEAK	128
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) ECDH x25519 (eq. 3072 bits RSA) FS WEAK	256
TLS_RSA_WITH_AES_128_GCM_SHA256 (0x9c) WEAK	128
TLS_RSA_WITH_AES_256_GCM_SHA384 (0x9d) WEAK	256
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f) WEAK	128
TLS_RSA_WITH_AES_256_CBC_SHA (0x35) WEAK	256
TLS_RSA_WITH_3DES_EDE_CBC_SHA (0xa) WEAK	112
# TLS 1.1 (suites in server-preferred order)	
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013) ECDH x25519 (eq. 3072 bits RSA) FS WEAK	128
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) ECDH x25519 (eq. 3072 bits RSA) FS WEAK	256
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f) WEAK	128
TLS_RSA_WITH_AES_256_CBC_SHA (0x35) WEAK	256
TLS_RSA_WITH_3DES_EDE_CBC_SHA (0xa) WEAK	112
# TLS 1.0 (suites in server-preferred order)	
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013) ECDH x25519 (eq. 3072 bits RSA) FS WEAK	128
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) ECDH x25519 (eq. 3072 bits RSA) FS WEAK	256
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f) WEAK	128
TLS_RSA_WITH_AES_256_CBC_SHA (0x35) WEAK	256
TLS_RSA_WITH_3DES_EDE_CBC_SHA (0xa) WEAK	112
(P) This server prefers ChaCha20 suites with clients that don't have AES-NI (e.g., Android devices)	

Risk Level:

Likelihood - 1

Impact - 2

Recommendations:

It is recommended to disable weak ciphers in TLS v1.0, v1.1 and v1.2 protocols.

3.4 (HAL-04) AUTOMATED TESTING RESULTS - INFORMATIONAL

Description:

Halborn used some automation tools to inspect the source code and deployed web services. **nikto** is an open source web server scanner that performs extensive testing against web servers for multiple items. It also checks for server configuration items such as the presence of multiple index files, HTTP server options, and will try to identify installed web servers and software. **njsscan** is a static application testing tool that can find insecure code patterns in js applications using a simple matching **libsast** pattern finder and the syntax-aware semantic code pattern search tool called **semgrep**.

Risk Level:

Likelihood - 1

Impact - 1

Results:

Nikto result:

```
- Nikto v2.1.5
-----
+ Target IP:      127.0.0.1
+ Target Hostname: localhost
+ Target Port:    3000
+ Start Time:     2021-10-28 23:58:58 (GMT-7)
-----
+ Server: No banner retrieved
+ Retrieved x-powered-by header: Express
+ Server leaks inodes via ETags, header found with file /, fields: 0xw/74b 0xFq86Tl2ondWDfhG1rp/IeI/neP4
+ The anti-clickjacking X-Frame-Options header is not present.
+ Uncommon header 'x-content-type-options' found, with contents: nosniff
+ Uncommon header 'content-security-policy' found, with contents: default-src 'none'
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ "robots.txt" retrieved but it does not contain any 'disallow' entries (which is odd).
+ Allowed HTTP Methods: GET, HEAD
+ 6544 items checked: 0 error(s) and 7 item(s) reported on remote host
+ End Time:      2021-10-28 23:59:16 (GMT-7) (18 seconds)
-----
+ 1 host(s) tested
```

Njsscan result:

```
- Pattern Match 255
- Semantic Grep 164

njsscan: v0.2.9 | Ajin Abraham | opensecurity.in
```

RULE ID	node_api_key								
OWASP	A3: Sensitive Data Exposure								
CWE	CWE-798: Use of Hard-coded Credentials								
DESCRIPTION	A hardcoded API Key is identified. Store it properly in an environment variable.								
SEVERITY	ERROR								
FILES	<table border="1"><tr><td>File</td><td>.env</td></tr><tr><td>Match Position</td><td>1 - 60</td></tr><tr><td>Line Number(s)</td><td>15</td></tr><tr><td>Match String</td><td>REACT_APP_INFURA_API_KEY="fdf88904416b4232864d3f75ba3d6947"</td></tr></table>	File	.env	Match Position	1 - 60	Line Number(s)	15	Match String	REACT_APP_INFURA_API_KEY="fdf88904416b4232864d3f75ba3d6947"
File	.env								
Match Position	1 - 60								
Line Number(s)	15								
Match String	REACT_APP_INFURA_API_KEY="fdf88904416b4232864d3f75ba3d6947"								

```
ethsec@ubuntu:~/0xnodes_new/platform/packages/app$
```

In the `njsscan` results, the hardcoded credentials discovered were already mentioned in `HAL-01 HARDCODED API KEY FOUND`.

THANK YOU FOR CHOOSING

// HALBORN