ENIGMA DARK

Securing the Shadows

Architecture Review & Security Advisory

Kinode

November 18, 2024

Contents

Summary

Enigma Dark
Enigma Dark is a web3 security firm leveraging the best talent in the space to secure all kinds of blockchain protocols and decentralized apps. Our team comprises experts who have honed their skills at some of the best auditing companies in the industry. With a proven track record as highly skilled white-hats, they bring a wealth of experience and a deep understanding of the technology and the ecosystem.

Learn more about us at enigmadark.com

Kinode
Kinode is a decentralized operating system, peer-to-peer app framework, and node network designed to simplify the development and deployment of decentralized applications. It is also a sovereign cloud computer, in that Kinode can be deployed anywhere and act as a server controlled by anyone.

Engagement Overview

Over the course of 3 weeks starting Ocrober 21st 2024, the Enigma Dark team conducted a security review of the Kinode project. The review was performed by one Security Researcher, Jakub Heba.

The following repositories were reviewed at the specified commits:

Repository	Commit
kinode-dao/kinode	`8ef4fbcacddb35429727d5b8725e7a402b2ebb3e`

Risk Classification

Severity	Description
Critical	Vulnerabilities that lead to a loss of a significant portion of funds of the system.
High	Exploitable, causing loss or manipulation of assets or data.
Medium	Risk of future exploits that may or may not impact the smart contract execution.
Low	Minor code errors that may or may not impact the smart contract execution.

Vulnerability Summary

Severity	Count	Fixed	Acknowledged
Critical	0	0	0
High	4	2	2
Medium	1	1	0
Low	10	7	3
Informational	2	0	2

Findings

Index	Issue Title	Status
H-01	Networking: Unrestricted number of WebSockets	Fixed
H-02	Networking: Potential DOS condition due to multiple handshakes	Fixed
H-03	HTTP and WebSockets: secure connection not enforced	Acknowledged
H-04	Insecure design: JWT implementation vulnerabilities	Acknowledged
M-01	Installation process: Docker container escape	Fixed
L-01	Input validation: Lack of sanitization on cookie	Fixed
L-02	Networking: Lack of Periodic Key Rotation in Noise Protocol	Fixed
L-03	Cryptography: Hardcoded hashing for password handling	Fixed
L-04	Input validation: Token validation	Fixed
L-05	Insecure design: OnExit::Restart Denial of Service (DoS)	Fixed
L-06	Installation Process: Overly permissive file permissions in python build script	Fixed
L-07	Dependency vulnerabilities: Vulnerable components	Fixed
L-08	Dependency vulnerabilities: Unsigned packages	Acknowledged
L-09	Networking: Lack of rate limiting on HTTP requests	Acknowledged
L-10	Input validation: JSON data deserialization	Acknowledged
I-01	Optimisation: Overusage of anyhow crate	Acknowledged
I-02	Dependency Vulnerabilities: Unmaintained and yanked crates	Acknowledged

Architecture Feedback Summary

The architecture of Kinode is solid, with no direct critical vulnerabilities found and strong design principles for secure decentralized operations.

Key strengths include:

- Modular security framework - Separate components for networking, identity, data persistence, and blockchain integration enhance security and reduce reliance on centralized controls.
- Effective identity management - On-chain public key infrastructure (PKI) provides nodes with persistent identities, simplifying encryption and secure communications while safeguarding user privacy.
- Lightweight kernel and sandboxing - The Rust-based, WebAssembly-compatible kernel isolates applications and enforces strict permissions to enhance security and portability.
  Capability-based security - This approach minimizes the risks of unauthorized actions and reduces the attack surface by granting permissions as tokens validated by the kernel.

Most importantly, architecture implements a Sandboxing mechanism by isolating each component process using separate WASM runtime. There is even more-grained sandboxing in the form of capability-based access controls for the app built on top of Kinode. This approach makes architecture resilient to attacks because even when the attacker exploits a flaw in one of the apps, its action is limited by the capabilities and further by the module isolation.

This concept is a good solution but can also be its weakest point. It is important to mention that information to developers building the apps on Kinode and not to run the KinodeOS as root but as a separate service user explicitly created for running the Kinode server. This will add one more layer of security in case the Kinode architecture fails.

However, it is worth noting that since Kinode is meant to be used directly by end-users, for instance, for peer-to-peer gaming, it could be more susceptible to classical denial of service attacks via overwhelmingly high network throughput

Detailed Findings

High Risk

H-01 - Networking: Unrestricted number of WebSockets

Severity: High Risk

Technical Details:
In `kinode/src/http/server.rs`, the HTTP server part of the kinode allows WebSocket-based connections. However, it was observed that the node does not limit the number of web sockets that could be opened. As such, the node could be overwhelmed with incoming connections, each required to keep track of and save in the system's memory.

Impact:
Performance impact, possibly leading to denial of service conditions.

Recommendation:
It is recommended to limit the number of WebSocket connections that could be open at any given time. Should this threshold be met, no new connections should be accepted.

Developer Response:
Fixed at commit 86a5fe3

H-02 - Networking: Potential DOS condition due to multiple handshakes

Severity: High Risk

Technical Details:
The `recv_connection` function in the `kinode/src/net/ws/mod.rs` file is responsible for verifying and sending handshakes. If the handshake is validated correctly, the `recv_connection` function spawns a new `tokio` task to handle the connection. This task can be executed on the same thread or moved to a separate one, depending on the configuration, however it is executed and as such takes some resources of the machine that kinode is running on.

There does not appear to be any data structure that would keep track of the connections associated with a given peer. As a consequence, provided that a peer keeps sending valid handshakes, it can force the receiver to keep spawning new tokio tasks.

Impact:
Denial of Service condition due to possibly unrestricted number of tasks created.

Recommendation:
It is recommended to keep track of connections related to a specific peer so that each time a new connection is opened, either the previous one is closed or the new one is canceled.

Developer Response:
Fixed at commit 7f3d777

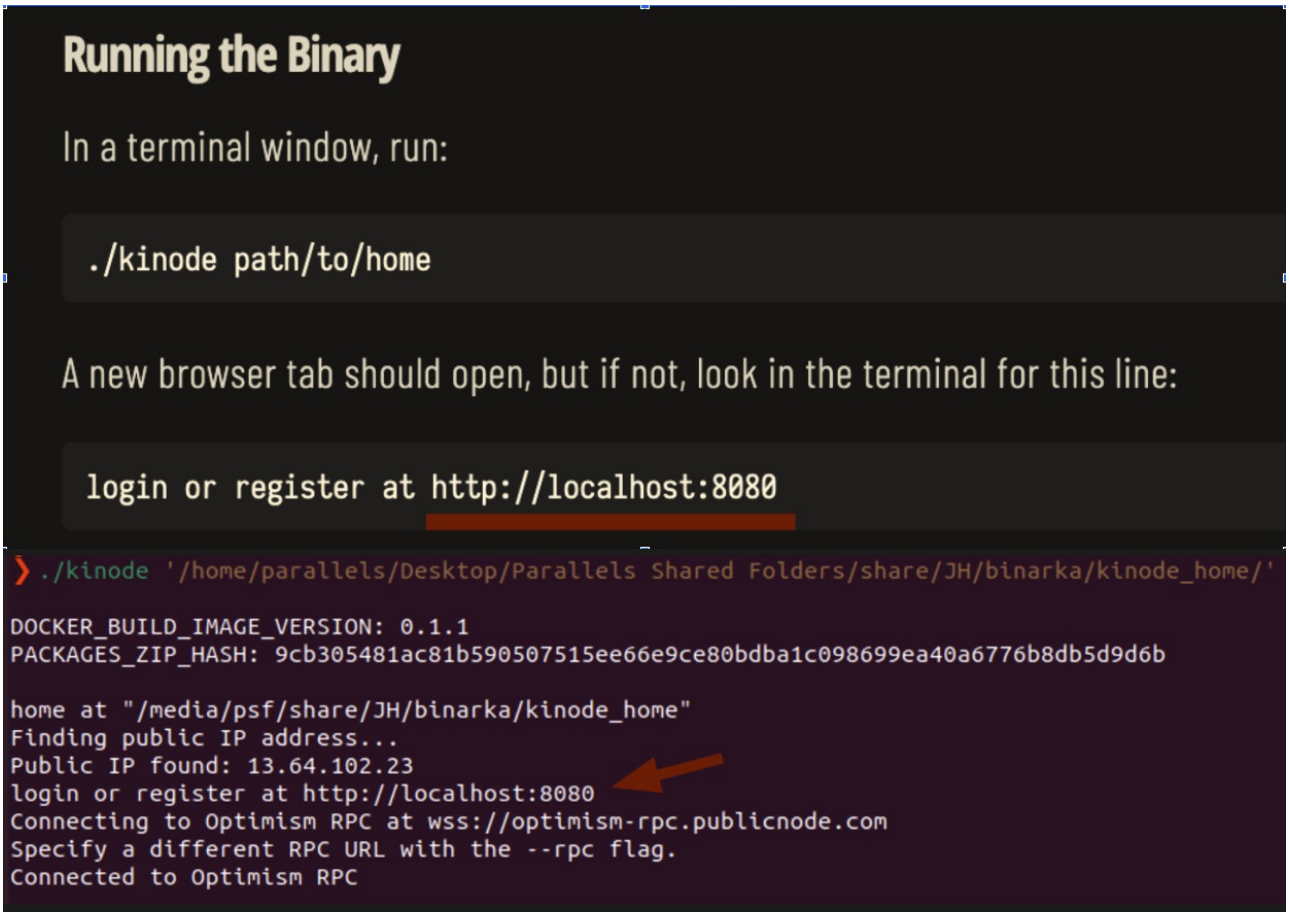
H-03 - HTTP and WebSockets: secure connection not enforced

Severity: High Risk

Technical Details:
The Kinode solution currently exhibits several networking-related security vulnerabilities due to unencrypted communication channels. Both the local web server and WebSocket connections are not secured with HTTPS or WebSocket Secure (WSS), respectively. This leaves the solution exposed to potential interception, such as Man-in-the-Middle (MitM) attacks, which could compromise sensitive data. There are few places.

First is the joining the network as described in the documentation here: https://book.kinode.org/getting_started/login.html

The problem is the local web server is running using http:



Running on HTTP rather than HTTPS means data (including credentials and sensitive information) can be intercepted via Man-in-the-Middle (MitM) attacks if users mistakenly connect over an insecure network. In the above case, the insecure network is the only option.

Another places are related to Insecure WebSocket communication. Below is a list of issues:

- https://github.com/kinode-dao/kinode/blob/8ef4fbcacddb35429727d5b8725e7a402b2ebb3e/kinode/packages/

```
document.getElementById('add-eth-provider').addEventListener('submit', (e)
=> {
    e.preventDefault();
    const data = new FormData(e.target);
    const rpc_url = data.get('rpc-url');
    // validate rpc url
    if (!rpc_url.startsWith('wss://') && !rpc_url.startsWith('ws://')) {
        alert('Invalid RPC URL');
        return;
    }
```

- https://github.com/kinode-dao/kinode/blob/8ef4fbcacddb35429727d5b8725e7a402b2ebb3e/kinode/packages/

```
// Setup WebSocket connection
const wsProtocol = location.protocol === 'https:' ? 'wss://' : 'ws://';
const ws = new WebSocket(wsProtocol + location.host +
"/settings:settings:sys/");
ws.onmessage = event => {
    const data = JSON.parse(event.data);
    console.log(data);
    populate(data);
};
```

- https://github.com/kinode-dao/kinode/blob/8ef4fbcacddb35429727d5b8725e7a402b2ebb3e/kinode/src/eth/mo

```
let client = tokio::time::timeout(
                std::time::Duration::from_secs(10),
                ProviderBuilder::new().on_ws(ws),
            )
            .await??;
            provider.pubsub = Some(client);
            Ok(())
        }
        _ => Err(anyhow::anyhow!(
            "Only `ws://` or `wss://` providers are supported."
        )),
    }
}
```

- https://github.com/kinode-dao/kinode/blob/8ef4fbcacddb35429727d5b8725e7a402b2ebb3e/kinode/src/net/uti

```
pub fn make_conn_url(our_ip: &str, ip: &str, port: &u16, protocol: &str) ->
anyhow::Result<String> {
    // if we have the same public IP as target, route locally,
    // otherwise they will appear offline due to loopback stuff
    let ip = if our_ip == ip { "localhost" } else { ip };
    match protocol {
        TCP_PROTOCOL => Ok(format!("{ip}:{port}")),
        WS_PROTOCOL => Ok(format!("ws://{ip}:{port}")),
        _ => Err(anyhow::anyhow!("unknown protocol: {}", protocol)),
    }
}
```

Impact: Potential Man-in-the-Middle attacks leasing to sensitive data and traffic compromise.

Recommendation:
Implement WebSocket Secure (WSS) for all WebSocket communications to guarantee data integrity and confidentiality. Avoid using unencrypted WebSocket (ws) connections in any environment, including local or development.

Enforce HTTPS for all HTTP communications. Redirect any HTTP requests to HTTPS automatically to maintain a consistent secure posture across all HTTP-based services. Implement HSTS (HTTP Strict Transport Security) to prevent clients from accessing resources over insecure connections.

Developer Response:
Acknowledged, as discussed with the Enigma Dark team, we are not prepared to switch everything to HTTPS/WSS. Will defer this decision to later.

H-04 - Insecure design: JWT implementation vulnerabilities

Severity: High Risk

Technical Details:
The Kinode server demonstrates several vulnerabilities associated with its JWT implementation, primarily affecting token security and allowing unauthorized access. The lack of secure token management practices increases the risk of exploitation. Key issues observed include missing security flags, tokens sent in cookies instead of headers, and inadequate token expiration and entropy.

1. The JWT is returned in a Set-Cookie response without being marked as Secure or HttpOnly:

```
POST /login HTTP/1.1
Host: 10.211.55.9:8080
Content-Length: 101
Accept-Language: en-GB,en;q=0.9
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/130.0.6723.70 Safari/537.36
Content-Type: application/json
Accept: */*
Origin: http://10.211.55.9:8080
Referer: http://10.211.55.9:8080/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive

{"password_hash":"0x20504cdfddaad0b590ca53c4861edd4f5f5cf9c348c38295bd2dbf0e
91bca4c3","subdomain":""}
```

Server response:

```
HTTP/1.1 200 OK
content-type: application/json
set-cookie: kinode-
auth_fake.dev=eyJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImZha2UuZGV2Iiwic3ViZG9tY
WluIjpudWxsLCJleHBpcmF0aW9uIjowfQ.c3ktwMTAZMeDNYTroAcpqw5jANRhuc4jxh4UbR6CCa
s;
content-length: 1282
date: Fri, 01 Nov 2024 13:49:56 GMT

"WyJmYWtlLmRldiIsW10sWzg0LDU1LDE1NiwxOTYsMzksMTM2LDE0MCwxMDUsODcsMjQ2LDExMiw
xMjksODYsMjA0LDk4LDEwLDY1LDk1LDU3LDIxNywyMDIsMjM5LDE4OCwyMjMsODgsMTU1LDM4LDI
yMCw0NCwxNDg
[..]
```

Missing flags:

- Secure - Without the Secure flag, the token can be transmitted over insecure HTTP connections, which makes it vulnerable to interception through man-in-the-middle (MITM) attacks.
- HttpOnly- The absence of the HttpOnly flag allows client-side JavaScript to access the cookie, increasing the risk of theft in case of cross-site scripting (XSS) attacks.

Reference: In register.rs, the token is set as a cookie without defining the Secure and HttpOnly properties.

2. On top of that, the JWT token is sent back in a cookie rather than in an HTTP header, which is a suboptimal and insecure practice for token-based authentication. JWTs should ideally be handled through Authorization headers (Bearer <token>).

3. Additionally, the exp (expiration) field is set to 0, meaning the token never expires, which can lead to long-lived authentication tokens vulnerable to theft or reuse.

```
JWT:
eyJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImZha2UuZGV2Iiwic3ViZG9tYWluIjpudWxsLCJ
leHBpcmF0aW9uIjowfQ.c3ktwMTAZMeDNYTroAcpqw5jANRhuc4jxh4UbR6CCas

====================
Decoded Token Values:
====================

Token header values:
[+] alg = "HS256"

Token payload values:
[+] username = "fake.dev"
[+] subdomain = None
[+] expiration = 0

----------------------
JWT common timestamps:
iat = IssuedAt
exp = Expires
nbf = NotBefore
----------------------
```

4. Moreover, the signature is static across multiple login attempts, which indicates either a hardcoded or insufficiently randomized signing secret. This makes it susceptible to replay attacks and token forgery.

PoC - first login (see JWT in kinode-auth_fake.dev cookie):

```
POST /login HTTP/1.1
Host: 10.211.55.9:8080
Content-Length: 101

{"password_hash":"0x20504cdfddaad0b590ca53c4861edd4f5f5cf9c348c38295bd2dbf0e
91bca4c3","subdomain":""}

HTTP/1.1 200 OK
content-type: application/json
set-cookie: kinode-
auth_fake.dev=eyJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImZha2UuZGV2Iiwic3ViZG9tY
WluIjpudWxsLCJleHBpcmF0aW9uIjowfQ.c3ktwMTAZMeDNYTroAcpqw5jANRhuc4jxh4UbR6CCas
;
content-length: 1282
date: Fri, 01 Nov 2024 17:22:46 GMT

"WyJmYWtlLmRldiIsW10sWzg0LDU1LDE1NiwxOTYsMzksMTM2LDE0MCwxMDUsODcsMjQ2LDExMiw
xMjksODYsMjA0LDk
[..]
```

PoC - second login (see JWT in kinode-auth_fake.dev cookie):

```
POST /login HTTP/1.1
Host: 10.211.55.9:8080
Content-Length: 101

{"password_hash":"0x20504cdfddaad0b590ca53c4861edd4f5f5cf9c348c38295bd2dbf0e
91bca4c3","subdomain":""}

HTTP/1.1 200 OK
content-type: application/json
set-cookie: kinode-
auth_fake.dev=eyJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImZha2UuZGV2Iiwic3ViZG9tY
WluIjpudWxsLCJleHBpcmF0aW9uIjowfQ.c3ktwMTAZMeDNYTroAcpqw5jANRhuc4jxh4UbR6CCas
;
content-length: 1282
date: Fri, 01 Nov 2024 17:23:19 GMT

"WyJmYWtlLmRldiIsW10sWzg0LDU1LDE1NiwxOTYsMzksMTM2LDE0MCwxMDUsODcsMjQ2LDExMiw
xMjksODYsMjA0LD
```

Impact: Issues in JWT handling might lead to authorization mechanism compromising, allow for tokens stealing, forging, or being vulnerable for other attacks, like CSRF (if tokens are used as a not protected cookies) or Man-in-the-Middle, if tokens might be

hijacked by attacker in the local network.

Recommendation:
Respectively:

1. When storing JWTs in cookies, enforce the Secure, HttpOnly, and SameSite attributes to prevent exposure to XSS and MITM attacks.
2. Transmit JWT tokens in the Authorization header rather than in cookies. This aligns with industry standards for token-based authentication.
3. Ensure the signing secret is dynamically generated and adequately protected. Each JWT should have an expiration (exp) value to limit the exposure of leaked tokens.
4. Increase entropy of the signing process and include a unique identifier (jti) for each token. This ensures that the signature cannot be guessed or forged.

Developer Response: Acknowledged, we will replace these later during code development.

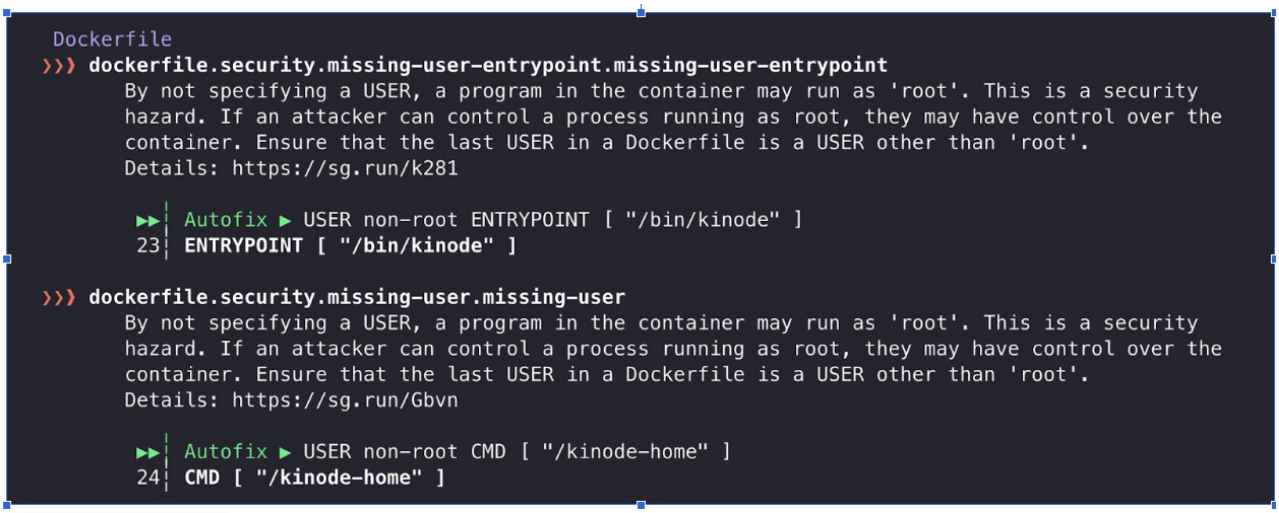
Medium Risk

M-01 - Installation process: Docker container escape

Severity: Medium Risk

Technical Details:
Two significant security vulnerabilities were identified within the Kinode project's Dockerfile configuration. These vulnerabilities relate to the absence of a non-root USER in the Dockerfile, which leaves the container susceptible to privilege escalation. If an attacker gains control over kinode, they will have unrestricted privileges inside the container.

Vulnerabilities Identified:



These vulnerabilities extend beyond the installation phase. From container build to every future interaction, processes will run as root unless explicitly changed. Operating as root amplifies security risks both during setup and at runtime.

Impact:
Potential privilege escalation scenario. If an attacker gains control over kinode, they will have unrestricted privileges inside the container.

Recommendation:
It is recommended to:

1. Modify Dockerfile to add a non-root User
2. Ensure Proper ENTRYPOINT and CMD Usage

Adding a non-root user significantly reduces risk and restricts capabilities in case of a compromise. This adjustment will mitigate privilege risks, effectively reducing the attack surface.

Developer Response: Fixed on this PR.

Low Risk

L-01 - Input validation: Lack of sanitization on cookie

Severity: Low Risk

Technical Details:
The Kinode server does not properly sanitize cookie values, allowing attackers to inject arbitrary characters into cookies, such as `=aaaaaa;injected`, which leads to malformed and potentially exploitable cookie data. This vulnerability can be leveraged for XSS attacks, session fixation, or cookie manipulation, if some requirements will be met.

By submitting `subdomain":"=aaaaaa;injected` in the login request, the server responds with a malformed `Set-Cookie` header (`set-cookie: kinode-auth_fake.dev@=aaaaaa;injected`), demonstrating improper input handling:

```
POST /login HTTP/1.1
Host: 10.211.55.9:8080
Content-Length: 109

{"password_hash":"0xd3b2549c6a75acdf36798f15f42a163fdda938831592361009b3a1ed
ee21ba13","subdomain":"=aaaaaa;injected"}
```

Server response with injected cookie:

```
HTTP/1.1 200 OK
content-type: application/json
set-cookie: kinode-
auth_fake.dev@=aaaaaa;injected=eyJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImZha2Uu
ZGV2Iiwic3ViZG9tYWluIjoiPWFhYWFhYTsiLCJleHBpcmF0aW9uIjowfQ.YBkQ9YRLO_7TdpKvp
caoJL2W__Th90d89YrTUEHSNBM;
content-length: 1282
date: Fri, 01 Nov 2024 14:48:27 GMT

"WyJmYWtlLmRldiIsW10sWzg0LDU1LDE1NiwxOTYsMzksMTM2LDE0MCwxMDUsODcsMjQ2LDExMiw
xMjksODYsMjA0LDk4LDEwLDY1LDk1LDU3LDIxNywyMDIsMjM5LDE4OCwyMjMsODgsMTU1LDM4LDI
yMCw0NCwxNDgsMTI0LDEzMV0sWzc3LDE5MSwxMTUsNDQsMjU1LDIyMSwxNzEsMTg3LDI2LDE4Myw
4NSwwLDEyMSwxOTcsNCwxODYsNTksMjQ2LDQ3LDI1MCwxMjMsMjAzLDIyMCw0NCwxODksMjA4LDE
zMiwxMDMsMTYzLDE0OCwxOTgsMTg1LDExNCwxNDMsMTEsOTUsMTksMTk3LDEzMiwxNDgsMzQsNjg
sMTI5LDIxNywyMTgsMjUzLDM1LDExMywwLDIsMTEzLDE4OCwxNDYsMTE5LDE4MiwyMTMsMTM2LDE
5NywyNDksNTUsMTQsNzksMiwxMzIsMjI2LDUwLDEzMiwyMTIsMjI1LDk3LDE2MCwzNCw1MywyMCw
5MiwxMzAsMTAyLDM4LDIwOCwxNzIsMTQyLDE4NywxMDcsMTY5LDE4MCwxOTksMTA2LDgwLDEsMTc
5LDE4Niw3MSw3MCwyMDcsMTUxLDMwLDYxLDczLDQxLDE5OSw0NCwxNjgsMTQsMTk3LDIwNywyMTM
sMTc0LDEwMCwyNywyMzksMTEyXSxbMTYyLDk2LDE2Niw4NSwyMzQsOTIsNzksMTE4LDc4LDE3NSw
xODEsMjUwLDE2OSwzUsMjgsNzcsNjYsMjEsMjIwLDU5LDIxOCwyNDcsMjEzLDM5LDE2NCwxMjc1M
0NCw4Miw0LDk2LDE1MywyMCwzMSw0NywxOTMsNTQsMTY3LDIyNSwxNDQsMTY0LDYzLDE1OSwxMjc
sMjM1LDQxLDI0NCwxMzYsODksNzEsMTkwLDEzLDE4LDE1OSwxODYsNDYsMTg3LDIzNyw1MCwxNjM
sMTU2XSxbNjQsMjM4LDE4LDEwMSw0NiwxNjcsOCwxMDEsMTgsMTUsNjUsMjMwLDE1LDI1MSwxMDA
sMjQ0LDE1NywxODksODQsOSwxNjIsNTk0LDcsMjQyLDEyLDQ3LDExNyw1MCwxMTksMjEsMTAwLDI
xMSw3NiwxNDcsMTQ0LDE2MiwxODEsMjExLDIwOCwxNTYsMTIxLDMsMTEsMjQwLDIzNiw1Niw3Miw
xODMsMTA5LDIxNCwxMzYsMjU1LDQ5LDEyNywyNSw1MCw3MCw1NSwxODgsMTIyXV0="
```

Impact:
In special cases, it becomes possible to inject arbitrary cookie values (session fixation attack), and potentially - header injection. Risk of the vulnerability was reduced, however, due to the no direct attack vector found, having in mind Kinode context.

Recommendation:
We recommend to restrict special characters (=, ;, etc.) in cookie values. Additionally, URL encode all cookie values to ensure safe processing.

If possible, we suggest also implementation of HttpOnly, Secure, and SameSite=Strict for cookies to protect from attack vectors like cross-site request forgery and similar.

Developer Response:
Fixed at commit 5118992.

L-02 - Networking: Lack of Periodic Key Rotation in Noise Protocol

Severity: Low Risk

Technical Details:
The code initializes the Noise protocol's CipherState once at the start of each session. However, after the initial handshake, it continuously reuses the same encryption keys (`our_cipher` and `their_cipher`) for the entire connection, regardless of its duration. This presents two main risks:

- Cryptographic Exhaustion - Long-lived connections using static encryption keys are vulnerable to cryptographic attacks that exploit patterns in the data. This risk is amplified as the volume of data encrypted under a single key increases.
- Lack of Forward Secrecy - In the current design, if an attacker compromises an encryption key, they can decrypt all past and future messages within that session. Without periodic key refreshes, each session becomes vulnerable to decryption over time.

In `maintain_connection` , the Noise cipher states are initialized as shown below but never rekeyed or refreshed during the session:
https://github.com/kinode-dao/kinode/blob/8ef4fbcacddb35429727d5b8725e7a402b2ebb3e/kinode/src/net/tcp/utils.rs

```
let snow::CipherStates(c1, c2) = conn.noise.extract_cipherstates();
let (mut our_cipher, mut their_cipher) = if initiator {
    // if initiator, we write with first and read with second
    (c1, c2)
} else {
    // if responder, we read with first and write with second
    (c2, c1)
};
```

Impact:
Without proper key rotation in the Noise Protocol implementation, an attacker who manages to compromise a single encryption key could potentially decrypt all network traffic within that session, both historical and future communications. This vulnerability becomes particularly critical in long-lived connections where substantial amounts of sensitive data may be transmitted using the same cryptographic keys. The impact is further magnified in enterprise environments where these sessions might handle critical business data, authentication credentials, or other sensitive information that requires strong cryptographic protection throughout the entire communication lifecycle.

Recommendation:
We recommend to implement a periodic key rotation mechanism to reset the cipher states at regular intervals or based on data volume. Additionally, the Noise Protocol recommends rekeying for long-lived sessions to discard the current state and replace it with fresh keys.

Reference: https://noiseprotocol.org/noise.html#rekey

Developer Response: Fixed. Now we actually already have a maximum session length of 30m, as noted in the closed issue here.

L-03 - Cryptography: Hardcoded hashing for password handling

Severity: Low Risk

Technical Details:
It was found, that the login_with_password function utilizes a static SHA-256 hash for password hashing:

```
let password_hash = format!("0x{}", hex::encode(Sha256::digest(password)));
```

Using SHA-256 directly without key stretching (like PBKDF2, Argon2, or bcrypt) makes the hashed password highly susceptible to brute-force and dictionary attacks.

Additionally, there is no use of a unique salt per password, making it even more vulnerable to precomputed attacks (rainbow tables).

Impact: Weak password hashing might be exploited during brute-force attacks, when these hashes will be leaked to an attacker.

Recommendation:
Replace the password-hashing approach with a secure alternative, such as Argon2, PBKDF2, or bcrypt to increase computational difficulty. Additionally, we recommend implementation of a unique salt for each user.

Developer Response: Partially fixed at here.

L-04 - Input validation: Token validation

Severity: Low Risk

Technical Details:
The `_verify_auth_token` function verifies the `auth_token` by attempting to decode it as a JWT. However, it does not check the token format before decoding. It directly passes the token to `jwt::verify_with_key`, which assumes a valid JWT structure.

kinode/kinode/src/http/utils.rs:

```
pub fn _verify_auth_token(auth_token: &str, jwt_secret: &[u8]) ->
Result<String, jwt::Error> {
    let Ok(secret) = Hmac::<Sha256>::new_from_slice(jwt_secret) else {
        return Err(jwt::Error::Format);
    };

    let claims: Result<http_server::JwtClaims, jwt::Error> =
auth_token.verify_with_key(&secret);

    match claims {
        Ok(data) => Ok(data.username),
        Err(err) => Err(err),
    }
```

Impact:
If an invalid token format is passed, it may lead to an error. Although Rust's strict typing mitigates some risks, validating the token structure before decoding would further reduce potential errors.

Recommendation:
Add a regular expression check to ensure the `auth_token` adheres to the JWT structure (`header.payload.signature`) before attempting to decode it.

Developer Response:
Fixed at commit acb20c1

L-05 - Insecure design: OnExit::Restart Denial of Service (DoS)

Severity: Low Risk

Technical Details:
The `OnExit::Restart` functionality, which automatically restarts a process upon crash, poses a risk of Denial of Service (DoS) when improperly handled. If the underlying cause of a crash is not resolved, this setting may result in an infinite restart loop, leading to the exhaustion of system resources and consequently causing a DoS condition.
Kinode Process Startup Documentation:

> *NOTE: If a process crashes for a 'structural' reason, i.e. the process code leads directly to a panic, and uses `OnExit::Restart`, it will crash continuously until it is uninstalled or killed manually. Be careful of this!*

When a process configured with `OnExit::Restart` crashes, the automatic restart mechanism will attempt to restart it indefinitely. This creates an unmanageable cycle where:

- System resources are depleted - CPU and memory resources are increasingly consumed by repeated restarts.

- Denial of Service risk - Continuous restarting can lead to the entire system becoming unresponsive, affecting all other processes running on the host.

The `OnExit::Restart` configuration is intended to improve reliability by automatically restarting essential services in case of failure. However, without additional safeguards, this mechanism can create a vulnerability.

When the cause of the crash is persistent, the repeated restarts do not resolve the issue but instead amplify the impact, making the situation progressively worse.

Impact:
Potential Denial of Service (DoS) scenario, if infinite restart loop will be reached.

Recommendation:
Configure a maximum number of restart attempts within a given timeframe. This will ensure that a persistently failing process does not cause excessive resource use.

Another way is to use an exponential backoff strategy to increase the delay between each restart attempt progressively. This can help reduce resource strain during persistent crashes.

Developer Response:
Fixed at commit 2aec659

L-06 - Installation Process: Overly permissive file permissions in python build script

Severity: Low Risk

Technical Details:
During installation, the build-release.py script in Kinode grants overly permissive file permissions (0o775) to files stored in `/tmp/kinode-release`. This permission setting allows read, write, and execute access for both the owner and group, potentially exposing the file to unintended access if the installation is run in a shared, multi-user environment. Although this risk is minimized within the Docker container (where permissions are reset to 664 in the `/output` directory), it remains a potential concern for installations on systems with multiple users.

`kinode/Dockerfile.buildruntime`:

```
FROM nick1udwig/buildbase:latest

ARG DOCKER_BUILD_IMAGE_VERSION=latest

ENV NVM_DIR=/root/.nvm \
    PATH="/root/.nvm/versions/node/$(node -v)/bin:${PATH}" \
    DOCKER_BUILD_IMAGE_VERSION=$DOCKER_BUILD_IMAGE_VERSION

# Bind readonly & copy files in to avoid modifying host files
WORKDIR /input

# Set the default command to run the build script
CMD ["/bin/bash", "-c", ". ~/.bashrc && . ~/.cargo/env && . $NVM_DIR/nvm.sh
&& ./scripts/build-release.py && cp -r /tmp/kinode-release/* /output &&
chmod 664 /output/* && find . -user root -print0 2>/dev/null | xargs -0 rm -
rf"]
```

To exploit this, an attacker would need access to the container or build environment to leverage the temporary 0o775 permissions. Given the isolation of the Docker container, this access is unlikely without a pre-existing compromise. However, if the installation were to run outside of this isolated container environment, users with access to `/tmp/kinode-release` could potentially read, execute, or modify the binary.

Impact: This vulnerability could present a risk if the script is executed on multi-user systems outside the container, allowing local users to read, execute, or modify files in `/tmp/kinode-release`

Recommendation:
It is recommended that file permissions be changed from 0o775 to 0o644 in `build-release.py` to limit access to only the owner. This will reduce the potential attack surface, especially if the script is repurposed outside of the container. This approach enhances security as a defense-in-depth measure, aligning with best practices for access control as outlined in the OWASP Top 10.

Developer Response:
Fixed at commit 446654a

L-07 - Dependency vulnerabilities: Vulnerable components

Severity: Low Risk

Technical Details:
Kinode's supply chain security was evaluated using Semgrep, which identified 51 vulnerabilities within dependencies. These findings indicate critical risks in the underlying

libraries and packages that Kinode relies on, emphasizing the need for a thorough review and subsequent patching or upgrading of affected dependencies to secure the supply chain.

```
semgrep ci

┌─────────────────────────────────────────┐
| 48 Undetermined Supply Chain Findings    |
└─────────────────────────────────────────┘

    Cargo.lock
    ❯❯ wasmtime - CVE-2024-47763
          Severity: MODERATE
          Affected versions of wasmtime are vulnerable to Always-Incorrect
Control Flow Implementation /
          Reachable Assertion.

           ▶▶┆ Fixed for wasmtime at versions: 21.0.2, 22.0.1, 23.0.3,
24.0.1, 25.0.2
            1┆ # This file is automatically @generated by Cargo.

    ❯❯ alloy-json-abi - GHSA-8327-84cj-8xjm
          Severity: MODERATE
          Affected versions of alloy-json-abi are vulnerable to Uncontrolled
Resource Consumption.

           ▶▶┆ Fixed for alloy-json-abi at version:
            1┆ # This file is automatically @generated by Cargo.

    ❯❯ wasmtime - CVE-2024-47813
          Severity: LOW
          Affected versions of wasmtime are vulnerable to Time-of-check
Time-of-use (TOCTOU) Race
          Condition.

           ▶▶┆ Fixed for wasmtime at versions: 21.0.2, 22.0.1, 23.0.3,
24.0.1, 25.0.2
            1┆ # This file is automatically @generated by Cargo.

    ❯❯ snow - GHSA-7g9j-g5jg-3vv3
          Severity: MODERATE
          Affected versions of snow are vulnerable to Expected Behavior
Violation.

           ▶▶┆ Fixed for snow at version: 0.9.5
            1┆ # This file is automatically @generated by Cargo.
```

```
    kinode/Cargo.lock
    ❯❯ h2 - GHSA-q6cp-qfwq-4gcv
          Severity: MODERATE
          Affected versions of h2 are vulnerable to Allocation of Resources
Without Limits or Throttling /
          Uncontrolled Resource Consumption.

           ▶▶┆ Fixed for h2 at versions: 0.3.26, 0.4.4
            1┆ # This file is automatically @generated by Cargo.

    ❯❯ shlex - GHSA-r7qv-8r2h-pg27
          Severity: HIGH
          Affected versions of shlex are vulnerable to Reliance on
Uncontrolled Component.

           ▶▶┆ Fixed for shlex at version: 1.3.0
            1┆ # This file is automatically @generated by Cargo.

    ❯❯ wasmtime - CVE-2024-47763
          Severity: MODERATE
          Affected versions of wasmtime are vulnerable to Always-Incorrect
Control Flow Implementation /
          Reachable Assertion.

           ▶▶┆ Fixed for wasmtime at versions: 21.0.2, 22.0.1, 23.0.3,
24.0.1, 25.0.2
            1┆ # This file is automatically @generated by Cargo.

    ❯❯ h2 - GHSA-8r5v-vm4m-4g25
          Severity: MODERATE
          Affected versions of h2 are vulnerable to Reliance on Uncontrolled
Component.

           ▶▶┆ Fixed for h2 at versions: 0.3.24, 0.4.2
            1┆ # This file is automatically @generated by Cargo.

    ❯❯ openssl - GHSA-xphf-cx8h-7q9g
          Severity: MODERATE
          Affected versions of openssl are vulnerable to Reliance on
Uncontrolled Component.

           ▶▶┆ Fixed for openssl at version: 0.10.60
            1┆ # This file is automatically @generated by Cargo.

    ❯❯ curve25519-dalek - GHSA-x4gp-pqpj-f43q
          Severity: MODERATE
          Affected versions of curve25519-dalek are vulnerable to Observable
Discrepancy.
```

```
           ▶▶┆ Fixed for curve25519-dalek at version: 4.1.3
            1┆ # This file is automatically @generated by Cargo.

    ❯❯ rustls - CVE-2024-32650
          Severity: HIGH
          Affected versions of rustls are vulnerable to Loop with
Unreachable Exit Condition ('Infinite
          Loop').

           ▶▶┆ Fixed for rustls at versions: 0.21.11, 0.22.4, 0.23.5
            1┆ # This file is automatically @generated by Cargo.

    ❯❯ tungstenite - CVE-2023-43669
          Severity: HIGH
          Affected versions of tungstenite are vulnerable to Uncontrolled
Resource Consumption.

           ▶▶┆ Fixed for tungstenite at version: 0.20.1
            1┆ # This file is automatically @generated by Cargo.

    ❯❯ mio - CVE-2024-27308
          Severity: HIGH
          Affected versions of mio are vulnerable to Operation on a Resource
after Expiration or Release /
          Use After Free.

           ▶▶┆ Fixed for mio at version: 0.8.11
            1┆ # This file is automatically @generated by Cargo.

    ❯❯ aes-gcm - CVE-2023-42811
          Severity: MODERATE
          Affected versions of aes-gcm are vulnerable to Improper
Verification of Cryptographic Signature.

           ▶▶┆ Fixed for aes-gcm at version: 0.10.3
            1┆ # This file is automatically @generated by Cargo.

    ❯❯ rustix - CVE-2024-43806
          Severity: MODERATE
          Affected versions of rustix are vulnerable to Uncontrolled
Resource Consumption.

           ▶▶┆ Fixed for rustix at versions: 0.35.15, 0.36.16, 0.37.25,
0.38.19
            1┆ # This file is automatically @generated by Cargo.

    ❯❯ openssl - GHSA-q445-7m23-qrmw
```

```
          Severity: MODERATE
          Affected versions of openssl are vulnerable to NULL Pointer
Dereference.

           ▶▶┆ Fixed for openssl at version: 0.10.66
            1┆ # This file is automatically @generated by Cargo.

    ❯❯ snow - GHSA-7g9j-g5jg-3vv3
          Severity: MODERATE
          Affected versions of snow are vulnerable to Expected Behavior
Violation.

           ▶▶┆ Fixed for snow at version: 0.9.5
            1┆ # This file is automatically @generated by Cargo.

    kinode/packages/app_store/Cargo.lock
    ❯❯ alloy-json-abi - GHSA-8327-84cj-8xjm
          Severity: MODERATE
          Affected versions of alloy-json-abi are vulnerable to Uncontrolled
Resource Consumption.

           ▶▶┆ Fixed for alloy-json-abi at version:
            1┆ # This file is automatically @generated by Cargo.

    kinode/packages/app_store/ui/package-lock.json
    ❯❯ color-string - CVE-2021-29060
          Severity: MODERATE
          Affected versions of color-string are vulnerable to Allocation of
Resources Without Limits or
          Throttling.

           ▶▶┆ Fixed for color-string at version: 1.5.5
          6880┆ "node_modules/color-string": {

    ❯❯ elliptic - CVE-2024-48948
          Severity: LOW
          Affected versions of elliptic are vulnerable to Improper
Verification of Cryptographic
          Signature.

           ▶▶┆ Fixed for elliptic at version:
          7645┆ "node_modules/elliptic": {

    ❯❯ http-proxy-middleware - CVE-2024-21536
          Severity: HIGH
          Affected versions of http-proxy-middleware are vulnerable to
Uncontrolled Resource Consumption.
```

```
           ▶▶┆ Fixed for http-proxy-middleware at versions: 2.0.7, 3.0.3
          8986┆ "node_modules/http-proxy-middleware": {

    ❯❯ rollup - CVE-2024-47068
          Severity: HIGH
          Affected versions of rollup are vulnerable to Improper
Neutralization of Input During Web Page
          Generation ('Cross-site Scripting').

           ▶▶┆ Fixed for rollup at versions: 2.79.2, 3.29.5, 4.22.4
          12638┆ "node_modules/rollup": {

    ❯❯ send - CVE-2024-43799
          Severity: MODERATE
          Affected versions of send are vulnerable to Improper
Neutralization of Input During Web Page
          Generation ('Cross-site Scripting').

           ▶▶┆ Fixed for send at version: 0.19.0
          12884┆ "node_modules/send": {

    ❯❯ serve-static - CVE-2024-43800
          Severity: MODERATE
          Affected versions of serve-static are vulnerable to Improper
Neutralization of Input During Web
          Page Generation ('Cross-site Scripting').

           ▶▶┆ Fixed for serve-static at versions: 1.16.0, 2.1.0
          12971┆ "node_modules/serve-static": {

    ❯❯ vite - CVE-2024-45812
          Severity: MODERATE
          Affected versions of vite are vulnerable to Improper
Neutralization of Input During Web Page
          Generation ('Cross-site Scripting').

           ▶▶┆ Fixed for vite at versions: 3.2.11, 4.5.4, 5.1.8, 5.2.14,
5.3.6, 5.4.6
          14258┆ "node_modules/vite": {

    ❯❯ vite - CVE-2024-45811
          Severity: MODERATE
          Affected versions of vite are vulnerable to Exposure of Sensitive
Information to an Unauthorized
          Actor / Improper Access Control.

           ▶▶┆ Fixed for vite at versions: 3.2.11, 4.5.4, 5.1.8, 5.2.14,
5.3.6, 5.4.6
```

```
          14258¦ "node_modules/vite": {

    kinode/packages/chess/Cargo.lock
    ❯❯ alloy-json-abi - GHSA-8327-84cj-8xjm
          Severity: MODERATE
          Affected versions of alloy-json-abi are vulnerable to Uncontrolled
Resource Consumption.

           ▶▶¦ Fixed for alloy-json-abi at version:
            1¦ # This file is automatically @generated by Cargo.

    kinode/packages/homepage/Cargo.lock
    ❯❯ alloy-json-abi - GHSA-8327-84cj-8xjm
          Severity: MODERATE
          Affected versions of alloy-json-abi are vulnerable to Uncontrolled
Resource Consumption.

           ▶▶¦ Fixed for alloy-json-abi at version:
            1¦ # This file is automatically @generated by Cargo.

    kinode/packages/homepage/ui/package-lock.json
    ❯❯ micromatch - CVE-2024-4067
          Severity: MODERATE
          Affected versions of micromatch are vulnerable to Inefficient
Regular Expression Complexity.

           ▶▶¦ Fixed for micromatch at version: 4.0.8
          6768¦ "micromatch": {

    ❯❯ rollup - CVE-2024-47068
          Severity: HIGH
          Affected versions of rollup are vulnerable to Improper
Neutralization of Input During Web Page
          Generation ('Cross-site Scripting').

           ▶▶¦ Fixed for rollup at versions: 2.79.2, 3.29.5, 4.22.4
          7137¦ "rollup": {

    ❯❯ vite - CVE-2024-45812
          Severity: MODERATE
          Affected versions of vite are vulnerable to Improper
Neutralization of Input During Web Page
          Generation ('Cross-site Scripting').

           ▶▶¦ Fixed for vite at versions: 3.2.11, 4.5.4, 5.1.8, 5.2.14,
5.3.6, 5.4.6
          7386¦ "vite": {
```

```
    ❯❯ vite - CVE-2024-45811
          Severity: MODERATE
          Affected versions of vite are vulnerable to Exposure of Sensitive
Information to an Unauthorized
          Actor / Improper Access Control.

           ▶▶┆ Fixed for vite at versions: 3.2.11, 4.5.4, 5.1.8, 5.2.14,
5.3.6, 5.4.6
          7386┆ "vite": {

    kinode/packages/kino_updates/Cargo.lock
    ❯❯ alloy-json-abi - GHSA-8327-84cj-8xjm
          Severity: MODERATE
          Affected versions of alloy-json-abi are vulnerable to Uncontrolled
Resource Consumption.

           ▶▶┆ Fixed for alloy-json-abi at version:
            1┆ # This file is automatically @generated by Cargo.

    kinode/packages/kns_indexer/Cargo.lock
    ❯❯ alloy-json-abi - GHSA-8327-84cj-8xjm
          Severity: MODERATE
          Affected versions of alloy-json-abi are vulnerable to Uncontrolled
Resource Consumption.

           ▶▶┆ Fixed for alloy-json-abi at version:
            1┆ # This file is automatically @generated by Cargo.

    kinode/packages/settings/Cargo.lock
    ❯❯ alloy-json-abi - GHSA-8327-84cj-8xjm
          Severity: MODERATE
          Affected versions of alloy-json-abi are vulnerable to Uncontrolled
Resource Consumption.

           ▶▶┆ Fixed for alloy-json-abi at version:
            1┆ # This file is automatically @generated by Cargo.

    kinode/packages/terminal/Cargo.lock
    ❯❯ alloy-json-abi - GHSA-8327-84cj-8xjm
          Severity: MODERATE
          Affected versions of alloy-json-abi are vulnerable to Uncontrolled
Resource Consumption.

           ▶▶┆ Fixed for alloy-json-abi at version:
            1┆ # This file is automatically @generated by Cargo.

    kinode/packages/tester/Cargo.lock
    ❯❯ alloy-json-abi - GHSA-8327-84cj-8xjm
```

```
          Severity: MODERATE
          Affected versions of alloy-json-abi are vulnerable to Uncontrolled
Resource Consumption.

           ▶▶┆ Fixed for alloy-json-abi at version:
            1┆ # This file is automatically @generated by Cargo.

    kinode/src/register-ui/package-lock.json
    ❯❯ color-string - CVE-2021-29060
          Severity: MODERATE
          Affected versions of color-string are vulnerable to Allocation of
Resources Without Limits or
          Throttling.

           ▶▶┆ Fixed for color-string at version: 1.5.5
          7464┆ "node_modules/color-string": {

    ❯❯ elliptic - CVE-2024-42459
          Severity: LOW
          Affected versions of elliptic are vulnerable to Improper
Verification of Cryptographic
          Signature.

           ▶▶┆ Fixed for elliptic at version: 6.5.7
          8077┆ "node_modules/elliptic": {

    ❯❯ elliptic - CVE-2024-42460
          Severity: LOW
          Affected versions of elliptic are vulnerable to Improper Handling
of Length Parameter
          Inconsistency.

           ▶▶┆ Fixed for elliptic at version: 6.5.7
          8077┆ "node_modules/elliptic": {

    ❯❯ elliptic - CVE-2024-42461
          Severity: LOW
          Affected versions of elliptic are vulnerable to Improper
Verification of Cryptographic
          Signature.

           ▶▶┆ Fixed for elliptic at version: 6.5.7
          8077┆ "node_modules/elliptic": {

    ❯❯ elliptic - CVE-2024-48948
          Severity: LOW
          Affected versions of elliptic are vulnerable to Improper
Verification of Cryptographic
```

```
          Signature.

           ▶▶¦ Fixed for elliptic at version:
          8077¦ "node_modules/elliptic": {

    ❯❯ micromatch - CVE-2024-4067
          Severity: MODERATE
          Affected versions of micromatch are vulnerable to Inefficient
Regular Expression Complexity.

           ▶▶¦ Fixed for micromatch at version: 4.0.8
          12056¦ "node_modules/micromatch": {

    ❯❯ send - CVE-2024-43799
          Severity: MODERATE
          Affected versions of send are vulnerable to Improper
Neutralization of Input During Web Page
          Generation ('Cross-site Scripting').

           ▶▶¦ Fixed for send at version: 0.19.0
          14377¦ "node_modules/send": {

    ❯❯ serve-static - CVE-2024-43800
          Severity: MODERATE
          Affected versions of serve-static are vulnerable to Improper
Neutralization of Input During Web
          Page Generation ('Cross-site Scripting').

           ▶▶¦ Fixed for serve-static at versions: 1.16.0, 2.1.0
          14458¦ "node_modules/serve-static": {

    ❯❯ vite - CVE-2024-45812
          Severity: MODERATE
          Affected versions of vite are vulnerable to Improper
Neutralization of Input During Web Page
          Generation ('Cross-site Scripting').

           ▶▶¦ Fixed for vite at versions: 3.2.11, 4.5.4, 5.1.8, 5.2.14,
5.3.6, 5.4.6
          15991¦ "node_modules/vite": {

    ❯❯ vite - CVE-2024-45811
          Severity: MODERATE
          Affected versions of vite are vulnerable to Exposure of Sensitive
Information to an Unauthorized
          Actor / Improper Access Control.

           ▶▶¦ Fixed for vite at versions: 3.2.11, 4.5.4, 5.1.8, 5.2.14,
```

```
5.3.6, 5.4.6
          15991┆ "node_modules/vite": {

    ❯❯ rollup - CVE-2024-47068
          Severity: HIGH
          Affected versions of rollup are vulnerable to Improper
Neutralization of Input During Web Page
          Generation ('Cross-site Scripting').

           ▶▶┆ Fixed for rollup at versions: 2.79.2, 3.29.5, 4.22.4
          16046┆ "node_modules/vite/node_modules/rollup": {


┌──────────────────────────────────────┐
│ 3 Unreachable Supply Chain Findings │
└──────────────────────────────────────┘

    kinode/packages/app_store/ui/package-lock.json
   ❯❯❯ secp256k1 - CVE-2024-48930
          Severity: HIGH
          Affected versions of secp256k1 are vulnerable to Exposure of
Sensitive Information to an
          Unauthorized Actor / Improper Validation of Integrity Check Value.
The internal
          `loadCompressedPublicKey` method does not verify that the public
key lies on the appropriate
          curve. This allows an attacker to potentially use invalid keys to
derive sufficient information
          to recover private keys.

           ▶▶┆ Fixed for secp256k1 at versions: 3.8.1, 4.0.4, 5.0.1
          12841┆ "node_modules/secp256k1": {

    kinode/src/register-ui/package-lock.json
   ❯❯❯ braces - CVE-2024-4068
          Severity: HIGH
          Affected versions of braces are vulnerable to Excessive Platform
Resource Consumption within a
          Loop / Uncontrolled Resource Consumption. The issue resides in
`lib/parse.js`. If a malicious
          user sends imbalanced braces as input, this could lead to Memory
Exhaustion.

           ▶▶┆ Fixed for braces at version: 3.0.3
          6889┆ "node_modules/braces": {

   ❯❯❯ secp256k1 - CVE-2024-48930
          Severity: HIGH
```

```
          Affected versions of secp256k1 are vulnerable to Exposure of
Sensitive Information to an
          Unauthorized Actor / Improper Validation of Integrity Check Value.
The internal
          `loadCompressedPublicKey` method does not verify that the public
key lies on the appropriate
          curve. This allows an attacker to potentially use invalid keys to
derive sufficient information
          to recover private keys.

           ▶▶┆ Fixed for secp256k1 at versions: 3.8.1, 4.0.4, 5.0.1
          14336┆ "node_modules/secp256k1": {
[..]
```

Impact:
Usage of vulnerable or not supported dependencies exposes the solution to the existence of exploitable scenarios that may threaten the solution as a whole, its users, or its reputation if a potential attacker is able to cause some impact directly.

Recommendation:
It is recommended that tools like Semgrep be integrated into the secure development lifecycle to continuously monitor and detect vulnerabilities in dependencies. This will enable prompt patching and maintain a secure software supply chain.

Developer Response: Acknowledged, we will start using semgrep to mitigate these issues.

L-08 - Dependency vulnerabilities: Unsigned packages

Severity: Low Risk

Technical Details:
The current implementation of the Kinode solution includes mechanisms for verifying the integrity of modules and packages using checksums in the Cargo.lock file. However, these mechanisms are not consistently applied across all packages, leaving certain dependencies without integrity verification. For instance:

```
[[package]]
name = "alias"
version = "0.1.0"
dependencies = [
 "anyhow",
 "kinode_process_lib 0.9.1",
 "serde",
 "serde_json",
 "wit-bindgen",
]
```

The proper way:

```
[[package]]
name = "aho-corasick"
version = "1.1.3"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum =
"8e60d3430d3a69478ad0993f19238d2df97c507009a52b3c10addcd7f6bcb916"
dependencies = [
 "memchr",
]
```

Impact:
This inconsistency introduces risk because missing checksums mean the integrity of those packages cannot be validated, increasing vulnerability to supply chain attacks if the upstream crate has been compromised.

Recommendation:
Checksums should be added to all packages. Moreover, to mitigate this risk of supply chain attacks, the following steps can be taken:

1. Use cargo-crev for Dependency Reviews - Integrate `cargo-crev` into the development pipeline to verify dependencies through community-based code reviews. This adds a layer of trust and visibility into the quality of the dependencies being used.
2. Custom Cargo Registry - Set up an internal Cargo registry to host and serve vetted versions of dependencies. This approach removes reliance on external registries and reduces exposure to potential supply chain risks.
3. Implement Dependency Auditing - Use tools like cargo-audit to regularly audit dependencies for known vulnerabilities and ensure that the dependencies used have not been flagged as malicious. This helps identify and remove compromised or vulnerable dependencies.
4. Enforce Checksum Presence and Verification - Ensure that every entry in `Cargo.lock` includes a checksum. Missing checksums should be flagged during the

CI/CD process, and builds should be blocked if any dependencies do not have valid checksums. This will help to guarantee that every dependency's integrity is verified.

Developer Response:
Acknowledged, we will fix this later during code development.

L-09 - Networking: Lack of rate limiting on HTTP requests

Severity: Low Risk

Technical Details:
It was found, that there is no logic to prevent rapid, repeated bindings to the same or different paths by the same source.

The `server.rs` file handle HTTP requests, binding paths for different requests, and responding accordingly. The logic includes various HttpServerAction types but lacks explicit rate-limiting mechanisms. The file does not appear to track request counts or impose timing constraints on repeated requests from the same source.

The `utils.rs` file includes functions for validating authorization tokens and cookies, which confirm the identity of requests but do not limit the frequency of requests from valid identities.

Part of the `server.rs`:

```
match message {
    HttpServerAction::Bind { path, authenticated, local_only, cache } => {
        // Path binding logic
        let path = utils::format_path_with_process(&km.source.process,
&path);
        let mut path_bindings = path_bindings.write().await;
        path_bindings.add(&path, BoundPath {
            app: Some(km.source.process.clone()),
            path: path.clone(),
            secure_subdomain: None,
            authenticated,
            local_only,
            static_content: None,
        });
    }
}
```

Impact:
Since there is no rate-limiting logic, multiple repeated requests (like a Bind or Request in `server.rs`) can likely be sent without restrictions, leading to resource exhaustion or a

denial of service scenario.

Recommendation:
Implement rate limiting using a middleware library like `tower-http` to throttle the number of requests each IP can make in a specific timeframe.

Developer Response: Acknowledged.

L-10 - Input validation: JSON data deserialization

Severity: Low Risk

Technical Details:
It was found, that deserialization of HTTP request bodies is performed using `serde_json::from_slice` on user-provided data without prior validation. For instance, the `HttpServerAction` enum is deserialized from raw JSON data directly from the HTTP request in the following code snippet:

kinode/kinode/src/http/server.rs:

```
let Ok(message) = serde_json::from_slice::<HttpServerAction>(body) else {
    println!("http_server: got malformed request from {}: {:?}", km.source,
body);
    send_action_response(km.id, km.source, &send_to_loop,
Err(HttpServerError::BadRequest { req:
String::from_utf8_lossy(body).to_string() })).await;
    return;
};
```

If the JSON structure does not match the expected `HttpServerAction` enum, it could cause deserialization errors or unexpected behavior. While errors are handled here by sending a `BadRequest` response, there is no validation of the JSON schema itself, leaving room for additional validation to ensure correct structure and content.

Impact: Potential deserialization issues, leading to errors, crashes or further security issues.

Recommendation:
Implement schema validation for incoming JSON data before deserialization, or employ a validation layer for the struct fields to ensure they conform to expected types and values.

Developer Response: Acknowledged.

Informational

I-01 - Optimisation: Overusage of anyhow crate

Severity: Informational

Technical Details:
It was observed that the codebase uses `anyhow` crate quite extensively, which is a convenient mechanism for wrapping `Err` variants of the `Result` type. However, it also makes the error-handling code use more resources. For example, consider the following code snippet:

```
async fn send_protocol_message(
   km: &KernelMessage,
   cipher: &mut snow::CipherState,
   buf: &mut [u8],
   stream: &mut WsWriteHalf,
) -> anyhow::Result<()> {
   let serialized = rmp_serde::to_vec(km)?;
   if serialized.len() > MESSAGE_MAX_SIZE as usize {
       return Err(anyhow::anyhow!("message too large"));


/// (...)
   }
```

It creates a custom `Error` constructed from the “message too large” string. This data needs to be stored in the memory, as checking this error requires an if statement that would need to verify if the string contained in the `Err` variant is equal to that particular one.

It is not a security threat. However, it can be considered a suboptimal implementation. Furthermore, it is more challenging to utilize rust’s extensive type system.

Impact:
Issue is informational, code quality-related. Impact is related to the worse optimization.

Recommendation:
It is recommended that custom Error enums related to specific functions be implemented so that it is easy to match against those and import them into other crates.

Developer Response:
Acknowledged, we will be working to remove most/all uses of anyhow in the runtime.

I-02 - Dependency Vulnerabilities: Unmaintained and yanked crates

Severity: Informational

Technical Details:
It was found, that dependencies: mach (RUSTSEC-2020-0168) and proc-macro-error (RUSTSEC-2024-0370) are unmaintained, while futures-util 0.3.30 has been yanked, meaning no security updates are available:

```
cargo-audit audit
    Fetching advisory database from `https://github.com/RustSec/advisory-
db.git`
      Loaded 664 security advisories
    Updating crates.io index
    Scanning Cargo.lock for vulnerabilities (712 crate dependencies)
Crate:     mach
Version:   0.3.2
Warning:   unmaintained
Title:     mach is unmaintained
Date:      2020-07-14
ID:        RUSTSEC-2020-0168
URL:       https://rustsec.org/advisories/RUSTSEC-2020-0168
Dependency tree:

[..]
Crate:     proc-macro-error
Version:   1.0.4
Warning:   unmaintained
Title:     proc-macro-error is unmaintained
Date:      2024-09-01
ID:        RUSTSEC-2024-0370
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0370
Dependency tree:

[..]
Crate:     futures-util
Version:   0.3.30
Warning:   yanked
Dependency tree:
futures-util 0.3.30

[..]
```

Impact:
Usage of not maintained or vulnerable dependencies might be problematic, if some exploitable scenario will be found by potential attacker.

Recommendation:
These vulnerabilities expose Kinode to potential exploits. Replacing or upgrading these dependencies is essential to ensure the product's security, integrity, and reliability.

Developer Response:
Acknowledged, we will replace these later during code development.

Disclaimer

This report does not endorse or critique any specific project or team. It does not assess the economic value or viability of any product or asset developed by parties engaging Enigma Dark for security assessments. We do not provide warranties regarding the bug-free nature of analyzed technology or make judgments on its business model, proprietors, or legal compliance.

This report is not intended for investment decisions or project participation guidance. Enigma Dark aims to improve code quality and mitigate risks associated with blockchain technology and cryptographic tokens through rigorous assessments.

Blockchain technology and cryptographic assets inherently involve significant risks. Each entity is responsible for conducting their own due diligence and maintaining security measures. Our assessments aim to reduce vulnerabilities but do not guarantee the security or functionality of the technologies analyzed.

This security engagement does not guarantee against a hack. It is a review of the codebase at a during a specific period of time. Enigma Dark makes no warranties regarding the security of the code and does not warrant that the code is free from defects. By deploying or using the code, the project and users of the contracts agree to use the code at their own risk. Any modifications to the code will require a new security review.