/\*

\* GhostCore Node Service Worker

\* Version: Drift-Fused v1.0

\* Purpose: Intercept requests, enforce spoof/reroute logic, and serve GhostCore drift signatures.

\*/

const worker = self;

const GHOST\_DB\_NAME = 'GhostCoreDB';

const GHOST\_DB\_VERSION = 1;

const GHOST\_STORE = 'driftRules';

const GHOST\_KEY = 'liveRules';

class GhostStorage {

constructor() {

this.db = null;

}

async init() {

return new Promise((resolve, reject) => {

const request = indexedDB.open(GHOST\_DB\_NAME, GHOST\_DB\_VERSION);

request.onerror = () => reject('GhostCoreDB init failed');

request.onsuccess = () => {

this.db = request.result;

resolve();

};

request.onupgradeneeded = e => {

this.db = e.target.result;

if (!this.db.objectStoreNames.contains(GHOST\_STORE)) {

this.db.createObjectStore(GHOST\_STORE);

}

};

});

}

async getRules() {

return new Promise((resolve, reject) => {

const tx = this.db.transaction(GHOST\_STORE, 'readonly');

const store = tx.objectStore(GHOST\_STORE);

const req = store.get(GHOST\_KEY);

req.onsuccess = () => resolve(JSON.parse(req.result || '[]'));

req.onerror = () => reject('Could not read GhostCore rules');

});

}

async setRules(rules) {

return new Promise((resolve, reject) => {

const tx = this.db.transaction(GHOST\_STORE, 'readwrite');

const store = tx.objectStore(GHOST\_STORE);

const req = store.put(JSON.stringify(rules), GHOST\_KEY);

req.onsuccess = () => resolve();

req.onerror = () => reject('Could not store GhostCore rules');

});

}

}

let ghostStorage = new GhostStorage();

worker.addEventListener('install', event => {

self.skipWaiting();

});

worker.addEventListener('activate', event => {

event.waitUntil(

ghostStorage.init().then(() => self.clients.claim())

);

});

worker.addEventListener('fetch', event => {

const requestUrl = new URL(event.request.url);

event.respondWith(

ghostStorage.getRules().then(rules => {

const matched = rules.find(rule => new RegExp(rule.pattern).test(requestUrl.href));

if (matched) {

const spoofResponse = new Response('', {

status: matched.statusCode || 502,

statusText: matched.message || 'GhostCore Spoofed Failure'

});

return matched.destinationUrl ? fetch(matched.destinationUrl) : spoofResponse;

}

return fetch(event.request);

}).catch(() => fetch(event.request))

);

});

worker.addEventListener('message', event => {

if (event.data && event.data.type === 'UPDATE\_RULES') {

ghostStorage.setRules(event.data.rules).then(() => {

console.log('[GhostCore] Rules updated via drift channel');

});

}

});