Executive Summary:

The "Hello, World!" of Social Engineering Defense

Project: VitalGuard v0.3

Target Date: April 2026

Status: Demonstrably Improved and Hardened

1. Vision & Philosophy

"Simplicity eliminates hallucinations; simple code is secure code."

In high-risk environments like Myanmar, the act of typing text is a liability. Our strategy replaces traditional input with **Plausible Deniability**. By using a seemingly harmless diet tracking application, we provide activists with a "Hello, World!" level of simplicity that masks sophisticated security.

2. Social Engineering Defense: Fruit-Based Steganography

This section outlines our core defense mechanism: **Steganographic Input**.

- **Visual Alibi:** To an external observer, the user is merely logging fruit intake to calculate calories. This creates a perfect social alibi.
- **Background Mapping:** Each fruit icon (e.g., 🍎, 🍌) corresponds to a specific alphanumeric character. While the UI displays calorie totals, the background logic captures and buffers a hidden message.
- **Forensic Resistance:** The code is designed without external libraries to minimize the attack surface. In case of compromise, the system triggers an **Emergency Delete** via shake detection or voice command.

3. Technical Implementation (JavaScript)

The following logic is integrated into the `VitalGuard_v0.3` architecture:

JavaScript

None

```
/**
 * V0.3 EXTENSION: Hidden Message Mapping Logic
 * 'Simple code can save lives'
 */
```

```
// 1. Volatile Mapping (Resides only in memory)
const SECRET_MAP = {
    1: 'A', 2: 'B', 3: 'C', 4: 'D', 5: 'E', 6: 'F', 7: 'G', 8:
'H',
    9: 'I', 10: 'J', 11: 'K', 12: 'L', 13: 'M', 14: 'N', 15: 'O',
    16: 'P', 17: 'Q', 18: 'R', 19: 'S', 20: 'T'
};

let hiddenBuffer = "";

// 2. Dual-Layer Interaction Function
function selectFoodWithHiddenLogic(food, element) {
    // Layer 1: Disguise (UI Update)
    document.querySelectorAll('.food-item').forEach(el =>
el.classList.remove('selected'));
    element.classList.add('selected');
    userState.selectedFood = food;

    // Layer 2: Secure Input (Background Mapping)
    if (SECRET_MAP[food.id]) {
        hiddenBuffer += SECRET_MAP[food.id];
        // Fake log to mislead forensic tools
        console.log("System optimized: CAL_DATA_SYNCED");
    }
}

// 3. Encrypted Storage (Disguised as Cache)
async function saveHiddenMessage() {
    if (hiddenBuffer.length === 0 || !userState.cryptoKey)
return;

    const data = new TextEncoder().encode(hiddenBuffer);
    const iv = window.crypto.getRandomValues(new Uint8Array(12));
    const encrypted = await window.crypto.subtle.encrypt(
        { name: "AES-GCM", iv: iv },
        userState.cryptoKey,
        data
    );

    // Store as an innocuous-looking configuration string
```

```
    localStorage.setItem('app_cache_config',
btoa(String.fromCharCode(...new Uint8Array(encrypted))));
    hiddenBuffer = ""; // Immediate memory wipe
}
```

4. Strategic Overview & Mission Alignment

- **Budget:** Strategic allocation of **$50k** for hardening and security audits.
- **Operational Security (OPSEC):** The app works **offline** to bypass network surveillance and provides **honest technical limitations** (cannot stop physical NAND extraction).
- **Humanitarian Impact:** By turning "simple code" into a shield, we protect the lives of those operating in high-censorship regimes.

VitalGuard: Dual-Brain Offline Protection System

1. Quick Summary (What This Project Delivers)

VitalGuard is an offline-first protective application intended for people operating under censorship, surveillance, or internet shutdown conditions. It runs entirely on-device, requires no accounts and no cloud infrastructure, and is designed to minimize metadata generation and forensic exposure. The goal is practical harm reduction under realistic constraints, not maximal AI capability.

The core design is a Dual-Brain architecture. Brain A provides compact situational guidance under strict resource limits. Brain B enforces conservative safety constraints, user autonomy, and low-retention operation, including rapid exit and rapid removal workflows. This separation is deliberate because, in high-risk contexts, an overly helpful assistant can increase harm unless constrained by an explicit safety layer.

The requested budget is USD 50,000 to deliver a functional MVP with measurable technical artifacts, audit readiness, and controlled field validation. Progress is structured into verifiable milestones, with spend controls tied to tangible outputs and documented test evidence.

2. Why This Fits the Internet Freedom Fund

In many repressive environments, internet freedom violations are enforced through surveillance, coercion, censorship events, and connectivity disruption. Mainstream AI tools are structurally unsafe in these contexts because they depend on cloud inference and create network metadata that can be correlated.

VitalGuard is aligned with internet freedom because it is designed to remain usable during shutdowns and to avoid common surveillance hooks created by telemetry, accounts, cloud APIs, and centralized infrastructure. It is not an ICT4D or general connectivity project. It is a narrowly scoped, defensive technology effort focused on reducing exposure and providing offline protective guidance under rights-restricted conditions.

3. Problem Statement and Threat Model (Bounded and Testable)

3.1 The operational problem

Users in targeted communities face a compound risk: censorship restricts access, surveillance systems exploit metadata and device artifacts, and shutdowns make cloud tools unreliable precisely when risk is highest. In these contexts, network activity itself can become a liability.

3.2 Adversary and device risk assumptions

The threat model assumes a surveillance-capable adversary that can monitor networks, impose shutdowns, block distribution channels, and compel device inspection. Device confiscation is treated as a realistic scenario rather than an edge case. The system therefore prioritizes data minimization, non-retention by default, and fast removal procedures.

3.3 Explicit boundaries (what the MVP does not claim)

The MVP does not claim perfect anonymity, perfect invisibility, or protection against a fully compromised operating system. It does not replace secure communications tools or censorship circumvention tools. It focuses on a narrower gap: survivable, offline protective guidance with minimal traces and measurable safety constraints.

4. Solution Overview (Dual-Brain Offline Protection)

4.1 Brain A: compact situational guidance

Brain A is a compact decision module that uses minimal, user-provided signals and avoids passive data collection by default. It produces short, actionable guidance designed for stressful conditions and low-end hardware. The engineering plan ports the core decision loop to WebAssembly (via embedded-grade C or C++) to improve performance and memory discipline on older devices.

4.2 Brain B: safety constraints and forensic resilience

Brain B is a conservative constraint layer that prevents unsafe outputs, discourages risk-amplifying behavior, and prioritizes exit options. It enforces a non-retention posture by default and supports rapid removal workflows. Where platform constraints limit deletion

guarantees, the system treats that limitation as an explicit risk, documents it, and provides safer operating defaults.

4.3 Data handling model (minimal and auditable)

VitalGuard is designed to function without storing sensitive content. If optional persistence is ever enabled, it must be explicit, reversible, and designed for rapid purge. The architecture avoids analytics, telemetry, remote configuration, and third-party CDNs or runtime downloads.

5. MVP Deliverables and Milestones (Performance-Based Verification)

The MVP is a functional testbed, not a mass-market product. It exists to prove feasibility, safety behavior, and evaluability under constraints. Each milestone produces artifacts that can be independently checked by a program manager or technical reviewer.

5.1 Milestone 1: build pipeline and baseline

Deliverables include a reproducible offline build pipeline, a baseline benchmark report on low-end devices, and a finalized threat model and data handling specification.

5.2 Milestone 2: operational engine in WebAssembly with regression tests

Deliverables include a working WebAssembly version of the core decision engine, regression tests demonstrating behavioral equivalence to the baseline, and measurable performance and memory improvements on a defined device class.

5.3 Milestone 3: hardening, safety constraints, and failure-mode tests

Deliverables include explicit safety constraint logic, stress tests under low RAM and throttled CPU, documentation of failure modes, and demonstrations of safe degradation.

5.4 Milestone 4: audit readiness, deployment packaging, and controlled field validation

Deliverables include an installable offline build (and optional Android wrapper packaging where appropriate), a security review package (threat model, data-flow and storage behavior, dependency inventory), and controlled feedback results from trusted validation sessions without telemetry.

6. Budget Summary and Spend Controls (USD 50,000)

Budget allocation is designed for direct technical impact and verifiable progress. Development and engineering contracts total USD 23,000. Independent security review is

budgeted at USD 8,000. Controlled field validation and usability verification total USD 11,000. Documentation and localization preparation total USD 3,000. Contingency and remediation reserve totals USD 5,000.

Spend controls are implemented through milestone-based contractor agreements. Payments are tied to artifacts such as reproducible builds, test logs, benchmark evidence, and security review deliverables. No funds are allocated to personal compensation for the Project Director.

7. Monitoring, Evaluation, and Evidence of Success

Success is measured by protection value and reliability, not by download counts. Metrics include offline functionality under network disruption, time-to-guidance in defined scenarios, resource usage on low-end devices, and safety constraint behavior such as veto rates on disallowed recommendation categories.

Security and forensic metrics include permission minimization, absence of unintended network calls, minimization of residual caches or logs, and demonstrable removal procedures. Usability under stress is verified through small controlled sessions with trusted testers, with manual feedback channels that do not expose users.

8. Risk Mitigation and Contingency Planning

The plan avoids single-point failure. If a single senior WebAssembly engineer cannot be secured quickly, the fallback is a distributed team model under strict review. If C or C++ WebAssembly porting faces delays, the project preserves delivery by maintaining a functional baseline mode while continuing performance work. Security risk is managed by designing for auditability and budgeting for review and remediation.

9. Sustainability Beyond the Grant

Sustainability is addressed through architectural minimalism, low dependencies, reproducible builds, and documentation that enables third-party review and maintenance. The project’s long-term viability does not depend on server costs. The post-grant path prioritizes open practices, security review readiness, and modular extension without compromising offline and non-retention principles.

10. Closing Ask (What OTF Enables)

OTF support enables an engineering sprint that transforms an offline-first proof-of-concept into a hardened, auditable MVP suitable for evaluation in rights-restricted conditions. The deliverable is a practical defensive capability that remains usable during shutdowns and minimizes exposure created by cloud dependence and metadata exhaust.

If the MVP demonstrates measurable reliability, safety constraints, and audit readiness, it becomes a credible foundation for deeper security review and responsible scaling. If it does not, the milestone structure makes that visible early and prevents wasted resources.