INTERNET-DRAFT
Intended status: Standards Track
September 16, 2025
Expires: March 16, 2026

R. Bouziane
SecRoot.io

OODA-HTTP: Adaptive Security Framework for HTTP Communications
draft-secroot-ooda-http-02

Abstract

This document defines OODA-HTTP, a protocol extension and adaptive security framework that applies the Observe-Orient-Decide-Act (OODA) loop to HTTP/HTTPS communications. Each HTTP request is transformed into an observation signal, a decision vector, and an opportunity for active defense.

OODA-HTTP extends the traditional 4-phase loop into 7 phases: Observe, Orient, Decide, Act, Memorize, Adjust, and Cooperate - allowing systems to evolve and adapt during communication. The protocol introduces a structured header, 'OODA-Action', that carries context-driven actions between endpoints.

OODA-HTTP offers compatibility with existing infrastructure (e.g., TLS, HTTP/2/3), while enabling resilience against classical, behavioral, and quantum threats. It supports inter-protocol cooperation with DOTS, STAMP, SIEMs, and behavioral inference engines such as SVDD and PSLPSO.

This document defines protocol behavior, header format, core logic, and interoperability strategy.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at https://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at https://www.ietf.org/shadow.html

This Internet-Draft will expire on January 18, 2026.

Table of Contents

1. Introduction

HTTP and HTTPS communications are increasingly targeted by sophisticated threats, including intelligent automation, behavioral evasion, and quantum-capable attacks. Traditional HTTPS, while providing encryption, relies on static pre-negotiated parameters and lacks mechanisms for real-time behavioral defense.

OODA-HTTP introduces an adaptive security layer built upon the OODA loop: Observe, Orient, Decide, and Act. Inspired by decision models in dynamic environments, this extension enables endpoints, proxies, and applications to assess context and respond defensively in real time.

Beyond the original loop, OODA-HTTP incorporates three additional phases: Memorize, Adjust, and Cooperate. These phases allow systems to learn from observed behavior, adapt their detection strategies, and collaborate with external agents such as DOTS controllers, SIEM platforms, and TLS telemetry analyzers.

Each HTTP request becomes more than a data exchange - it is treated as a semantic signal, a decision vector, and a potential reflex trigger. OODA-HTTP transforms passive transport into an intelligent, self-defending protocol layer.

This document defines the architecture, phases, message formats, use cases, and integration models required to deploy OODA-HTTP in modern HTTP/HTTPS infrastructures.

2. Terminology

OODA Loop: A four-phase decision model: Observe, Orient, Decide, Act. Originally from military strategy, now applied to cybersecurity and protocol logic.

OODA-HTTP: An extension of HTTP that enables contextual decision-making through adaptive security headers, behavioral memory, and coordination with external agents. The OODA-HTTP engine implements a seven-phase loop.

OODA-Action: A structured HTTP header used to transmit threat context, scores, and defensive actions. Replaces the deprecated X-prefixed variant.

Threat Score: A numeric risk indicator (0-100) assigned to a session or request based on contextual analysis.

Temporal Memory: A mechanism to track behaviors, entropy, and anomalies across multiple requests over time.

Semantic Vector: A structured representation of a session’s metadata,

timing, and behavioral fingerprint. Used to detect behavioral drift or identity replays.

SVDD: Support Vector Data Description - A model to learn the boundary of normal behavior and detect anomalies.

PSLPSO: Polynomial Self-Learning Particle Swarm Optimization - A metaheuristic algorithm to optimize model hyperparameters.

STAMP: System-Theoretic Accident Model and Processes - A framework to understand root causes of systemic or behavioral failures.

Cooperative Defense: Cross-agent sharing of security context, trust levels, and adaptive intentions.

Fiche: A lightweight execution module in the OODA-HTTP engine. Each fiche implements one function in the OODA loop (e.g., scoring, memory, orientation). Fiches can be composed, replaced, or versioned independently.

Behavioral Identity Trace: A persistent, cumulative profile of a client’s session behavior across time. Used by Memorize and Adjust phases to detect evasive patterns or delayed attacks.

OODA Registry: A JSON-based registry of available actions, scores, categories, and policy mappings used by Decide and Act phases.

Vector Engine: A core component of the Orient phase responsible for constructing, comparing, and storing semantic vectors associate

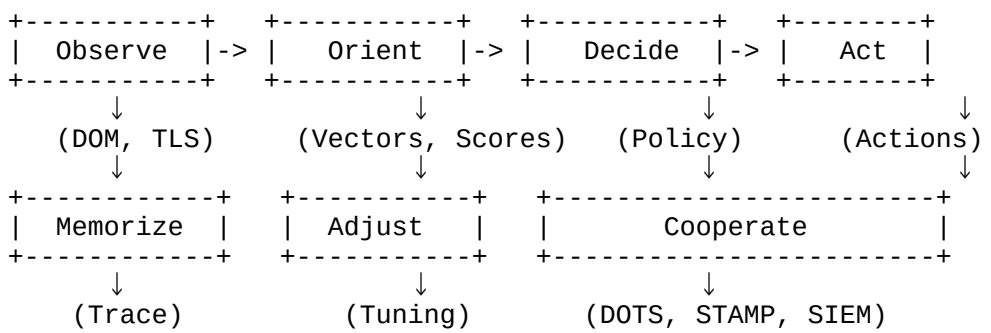
3. Architectural Overview

OODA-HTTP introduces an adaptive security layer within HTTP/HTTPS communication flows. It transforms each HTTP request into a semantic event processed through a modular decision loop based on the OODA model.

At its core, the architecture extends the traditional 4-phase OODA loop (Observe, Orient, Decide, Act) into a 7-phase reflex loop:
Observe -> Orient -> Decide -> Act -> Memorize -> Adjust -> Cooperate

This extension equips HTTP servers and proxies with real-time self-defensive capabilities, transforming them into intelligent agents capable of detecting, responding to, and learning from behavioral threats.

The OODA-HTTP engine is organized into modular functional blocks, each aligned with a phase of the loop. These modules can be deployed individually or as a complete pipeline, depending on operational needs.



Key architectural components include:

- **Observe:** Captures incoming request signals such as HTTP headers, user-agent strings, TLS handshake metadata, request frequency, and temporal patterns. At the frontend, DOM instrumentation (OODA-DOM) can

contribute additional behavioral data (e.g., typing patterns, interaction anomalies, tab focus).

- **Orient:** Processes observed signals to construct a contextual trace. This includes semantic enrichment, memory correlation, and optional embedding into vector space. Orientation is supported by scoring models such as SVDD, with dynamic tuning via PSLPSO.

- **Decide:** Applies decision logic based on thresholds, risk profiles, and detection rules. Decisions may include classifying the request as trusted, suspicious, automated, abusive, or unknown.

- **Act:** Executes the selected action in real time. Possible actions include forwarding, delaying, rejecting, annotating (e.g., tagging with a ‘OODA-Action‘ header), or applying countermeasures such as key rotation, CAPTCHA enforcement, or entropy tests.

- **Memorize:** Stores persistent behavioral identities, vector traces, and session-based scores. This phase enables the engine to evolve contextually by maintaining memory of request trajectories across time.

- **Adjust:** Continuously refines detection models and thresholds based on observed behavior, feedback loops, and temporal patterns. This enables adaptive scoring and risk management tailored to environments.

- **Cooperate:** Interfaces with external protocols and infrastructure such as DOTS (RFC 9132), STAMP, TLS proxies, and SIEM platforms. This allows sharing of insights, coordination of defense posture, and propagation of awareness across distributed systems.

Each functional block is implemented as a discrete module (called a "fiche") that can be audited, configured, and composed to fit deployment-specific requirements.

OODA-HTTP is designed to be:

- **Modular:** Each phase can operate independently or as part of a unified engine. - **Lightweight:** Minimal performance overhead by embedding logic at request time. - **Composable:** Deployable on the edge, server-side, or client-side. - **Backwards-compatible:** Fully compatible with HTTP/1.1, HTTP/2, and interoperable with HTTP/3 and QUIC.

This architecture enables OODA-HTTP to act as a reflexive overlay that enhances HTTP/HTTPS with native self-defense mechanisms, without centralized orchestration or cloud-based inspection.

The transition from the classic OODA model to the 7-phase OODA-HTTP loop is not purely theoretical: it is operationalized via concrete, testable modules and designed for real-time execution.

The following section (Section 4) details the extended loop and its role in HTTP-native decision-making.

4. The OODA Loop and Its Extension in HTTP

The OODA loop (Observe, Orient, Decide, Act) was originally developed as a decision-making framework in military contexts to describe how agents can respond dynamically to evolving situations. In the context of network communications, this loop provides a compelling foundation for adaptive defense mechanisms.

OODA-HTTP implements and extends this model directly within the HTTP processing stack, transforming each HTTP request into both a signal and a decision point. This allows servers to evaluate client behavior in real time, reason over context, and take action without depending on

external rule sets or cloud-based analysis layers.

The classical four phases of the OODA loop are:

- Observe: Gather input from the environment (request headers, payloads, timing, client context). - Orient: Analyze and interpret the input in light of past behavior, session history, and semantic meaning. - Decide: Select the most appropriate action or classification based on current context, scores, and thresholds. - Act: Execute the decision (e.g., forward, delay, redirect, reject, log, annotate).

OODA-HTTP extends the loop with three additional phases to enable persistence, learning, and collaboration:

- Memorize: Persist historical behaviors, request traces, and scores into a temporal memory structure. - Adjust: Update scoring models, detection thresholds, or feature weightings in light of evolving behavior patterns or newly observed anomalies. - Cooperate: Exchange contextual insights or warning signals with external agents such as DOTS clients, XDR platforms, or security orchestration systems.

This extended model yields a 7-phase reflex loop:
Observe -> Orient -> Decide -> Act -> Memorize
-> Adjust -> Cooperate

It reflects a living system - one that learns, adapts, and collaborates across the lifecycle of HTTP communications. In this paradigm, each HTTP request becomes not only a transport unit, but also a semantic event, contributing to a persistent, evolving understanding of network behavior.

Unlike abstract models, OODA-HTTP is built on an operational engine, composed of discrete and modular components. Each phase is powered by dedicated functional units, documented through a structured series of engineering fiches.

For example:

- The Observe phase is implemented through modules that capture request signals, client behavior patterns, and session dynamics. - The Orient phase is supported by trace analyzers, vectorization routines, and contextual enrichers. - The Decide and Act phases are handled by embedded rule engines and response controllers. - Memory and Adaptation are governed by evolving state management structures and score feedback loops. - Cooperation is enabled via connectors to external telemetry systems and threat intelligence relays.

Each of these components is designed as a fiche - a standardized, auditable, and modular building block. Together, they allow the OODA-HTTP engine to operate incrementally and autonomously across request flows.

OODA-HTTP does not merely defend. It evolves - through embedded logic, adaptive behavior, and cooperative defense.

As OODA-HTTP redefines the HTTP processing model through an extended loop of adaptive decision-making, it is essential to clarify how this approach aligns with and complements existing Internet protocols.

The following section outlines the positioning of OODA-HTTP in relation to established standards such as TLS, DOTS, and STAMP, highlighting how OODA-HTTP does not replace but extends their capabilities with real-time, request-level reflexes.

4.1 Observe Phase

The Observe phase is the initial entry point of the OODA-HTTP loop. It is responsible for capturing all available signals from the incoming HTTP request and its environment.

Observation in OODA-HTTP goes beyond traditional packet inspection or header parsing. It includes:

- Request metadata (method, path, headers, size, timing) - Session continuity (cookies, tokens, behavioral flow) - Temporal patterns (burst frequency, idle gaps, retries) - Frontend signal injection (via OODA-DOM where applicable)

These observations are collected in a structured behavioral trace, forming the basis for subsequent semantic analysis.

This phase is powered by a series of sensor components (documented as engineering fiches) that specialize in signal capture, logging, rate tracking, and correlation across requests and sessions.

The output of this phase is a normalized context trace, which is forwarded to the Orientation phase.

4.2 Orient Phase

The Orient phase processes the observed data and transforms it into meaningful behavioral context.

This step involves contextualization, enrichment, and preparation of features for decision-making. It includes:

- Mapping request patterns to known behavioral classes - Correlating current data with past traces (temporal context) - Building semantic vectors representing client behavior - Scoring interactions based on configurable heuristics

Orientation acts as the analytical core of the loop, transforming raw input into actionable knowledge. In doing so, it draws from both local memory (recent requests) and persistent memory (long-term scores).

Multiple engine components contribute to this phase: vector analyzers, scoring modules, memory readers, and enrichment agents - each defined as an auditable fiche within the OODA-HTTP framework.

The resulting oriented context is then passed to the Decide phase for evaluation.

4.3 Decide and Act Phases

The Decide phase evaluates the oriented behavioral context produced in the previous step. It is responsible for selecting an appropriate course of action based on scoring thresholds, classification models, and response policies.

Decision-making can be based on:

- Request-level scores (e.g., behavioral anomaly, session risk) - Rule matches (e.g., pattern detection, blacklists, policy triggers) - Historical trace evaluation (e.g., rate escalation, score increase)

The Act phase then enforces the selected action in response to the HTTP request. Examples include:

- Forwarding the request as-is - Modifying headers (e.g., adding ‘X-OODA-Action‘ or marking risk levels) - Delaying or throttling the response - Dropping or rejecting the request with context-aware messages

- Logging, alerting, or triggering external events

These two phases operate in tight succession and are implemented by action-oriented fiches such as scoring dispatchers, route switchers, or adaptive throttling modules.

Crucially, the decision logic is both local and dynamic. No central server or external orchestration is required, enabling real-time responsiveness within the HTTP processing path.

4.4 Positioning of OODA-HTTP in Relation to Existing Protocols

OODA-HTTP introduces a reflex loop directly into the HTTP stack, augmenting each request with local intelligence and adaptive behavior. While its capabilities are unique, its role is complementary to existing protocols rather than competitive or substitutive.

- Compared to TLS: TLS ensures encryption and integrity. OODA-HTTP operates above TLS, focusing on behavioral analysis and decision- making after decryption - without modifying the TLS layer itself.

- Compared to DOTS (DDoS Open Threat Signaling): DOTS provides centralized volumetric attack signaling. OODA-HTTP adds per-request defense logic, enabling instant response to stealthy or application-layer threats.

- Compared to STAMP and performance monitoring protocols: STAMP provides metrics for observability. OODA-HTTP integrates such metrics into a decision-making framework for threat mitigation.

By embedding real-time reflexes within the HTTP communication path, OODA-HTTP extends the Internet protocol toolbox with a semantic, session-aware, and autonomous layer of defense.

As such, it is best understood not as a replacement, but as a complement - enabling a new class of protocols where the server becomes an active, decision-capable agent in its own security posture.

5. Message Format and Protocol

OODA-HTTP introduces a structured HTTP header named ’OODA-Action’. This header conveys threat scores, classification categories, and recommended mitigation actions between client, server, edge proxy, or security middleboxes.

The header enables runtime decisions to be communicated across the HTTP lifecycle, and supports both passive evaluation and active intervention based on local or distributed policies.

Example:

OODA-Action: { "score": 82, "category": "suspicious", "action": "challenge", "timestamp": "2025-07-06T20:32:10Z" }

Fields:

score (integer): A numeric value between 0 and 100 indicating the perceived threat level associated with the request or session. Scores above configurable thresholds may trigger defensive actions.

category (string): A human-readable classification of the context. Typical values include ’normal’, ’suspicious’, or ’malicious’.

action (string): A recommended security response. Supported values include: * allow - Permit the request normally. * block - Reject the request immediately. * throttle - Introduce latency or rate

limits. * challenge - Request additional proof (e.g., CAPTCHA, MFA).

timestamp (string): ISO 8601 formatted UTC time of decision generation. Used for synchronization, log correlation, and caching logic.

This header may be generated by the server, an edge gateway, a CDN node, or even the browser (in OODA-DOM deployments). It can also be inserted or modified by intermediary security appliances as long as trust models permit.

Implementations SHOULD favor structured JSON for extensibility and clarity. In constrained environments, compact binary or base64-encoded variants MAY be used, provided semantic equivalence and cross-version compatibility.

Additionally, OODA-Action headers can be layered: - A client may send an initial OODA-Action (e.g., from DOM fingerprinting). - A proxy may append policy-based decisions (e.g., via reverse proxy logic). - A server may compute a final OODA-Action based on telemetry fusion, behavioral memory, or external coordination (e.g., DOTS, STAMP).

All OODA-Action headers MUST preserve traceability by including metadata about the originating component, when multiple evaluations are performed. This is essential to support cooperative defense and auditability.

Example with layered context:

OODA-Action: { "score": 92, "category": "malicious", "action": "block", "origin": "edge-proxy-eu3", "timestamp": "2025-07-06T20:32:10Z", "confidence": "high" }

Future extensions MAY define additional fields such as: - confidence levels - sub-scores per model - mitigation intents - trust assertions or cryptographic proofs

The structure of the header and its registry of values are detailed in Appendix B (Unified OODA Action Registry).

6. Threat Models and Detection

OODA-HTTP is designed to address multiple categories of threats across both application-level behaviors and lower-layer protocol interactions. Its 7-phase architecture enables continuous observation, evaluation, and response during HTTP communication.

6.1. Classical Threats

These include protocol-level and infrastructure attacks commonly addressed by traditional firewalls and rate-limiters, but now integrated into adaptive HTTP logic:

- DoS / DDoS floods (volumetric or protocol-based) - Slowloris and low-rate TCP connection exhaustion - Header manipulation (e.g., spoofed User-Agent, forged cookies) - Session replay and token/cookie theft

OODA-HTTP provides real-time scoring and behavioral fingerprinting, allowing precise throttling or blocking even under polymorphic attacks.

6.2. Behavioral Threats

Modern threats increasingly mimic legitimate users. OODA-HTTP detects deviations through session memory and pattern analysis:

- Irregular navigation paths or rage clicks - Bots rotating User-Agent and avoiding Referer headers - High-frequency scraping or burst access

patterns - Timing anomalies or inconsistent cookie behavior

The engine uses vector signatures and temporal memory to differentiate real users from automated threats - even when content and headers appear valid.

6.3. Quantum-Related Threats

While quantum computers are not yet mainstream, future-proofing is critical. OODA-HTTP monitors entropy and handshake quality to detect or deter:

- Low-entropy handshakes vulnerable to Grover-based attacks - Poor randomization in TLS session key generation - Delayed decryption scenarios leveraging Shor’s algorithm

Adaptive defenses include entropy reinforcement, dynamic TLS re-negotiation, and integration with post-quantum-ready entropy sources.

6.4. Cross-Layer Interoperability

To strengthen situational awareness, OODA-HTTP integrates with external defense systems such as DOTS (DDoS Open Threat Signaling) and STAMP (System-Theoretic Accident Model and Processes).

This integration allows:

- Exchange of early-warning signals and mitigation requests (via DOTS push) - Coordination with TLS/QUIC layers for handshake inspection - Feedback loops between application scoring and network-layer telemetry

For example, a DOTS signal indicating volumetric stress can influence OODA-HTTP’s decision to throttle or challenge even benign requests under pressure.

6.5. Layer Mapping

OODA-HTTP spans multiple OSI layers:

- Layer 3-4: Collaborates with DOTS, TLS/QUIC, and firewall metadata - Layer 7: Analyzes HTTP headers, cookies, timing, DOM events - Cross-layer: Reacts to feedback from SIEM, ML detectors, or edge proxies

This multi-layered detection strategy ensures coordinated responses without fragmenting the defense logic. It also allows fine-grained decision-making at each layer while maintaining a unified threat context.

7. Applications and Case Studies

OODA-HTTP has been tested across a variety of deployments - edge, proxy, gateway, and browser - to illustrate how its 7-phase loop adapts in practice. Each case demonstrates how HTTP requests become decision vectors for defense.

7.1. CDN Burst Detection

A CDN deployed OODA-HTTP at the edge layer to mitigate flash crowd events and bot-driven bursts. Using the Observe and Decide phases, traffic flow was scored in real time.

Sample outcome: OODA-Action: { "score": 74, "category": "burst", "action": "throttle" }

This action was propagated back to the client via the edge, applying rate controls without breaking session continuity.

7.2. Reverse Proxy Slowloris Defense

In a reverse proxy setup, the Memorize phase tracked slow socket behaviors and incomplete headers. Orient detected timing anomalies, and Decide triggered a block action for connection exhaustion attempts.

Sample outcome: OODA-Action: { "score": 91, "category": "low-rate", "action": "block" }

7.3. Bot Detection at the Application Gateway

A web application integrated a semantic vector engine using SVDD + PSLPSO. Behavioral inconsistencies across navigation paths were detected. OODA-HTTP inferred bot camouflage (rotating User-Agent + no Referer) and challenged the user.

Sample outcome: OODA-Action: { "score": 88, "category": "bot-camouflage", "action": "challenge" }

7.4. Quantum Resilience Testbed

In a TLS 1.3 testbed, OODA-HTTP monitored entropy quality during session negotiation. When entropy from the client nonce appeared predictable,OODA-HTTP suggested secret rotation through internal memory flags.

This scenario demonstrates the engine’s Adjust phase in action - without modifying TLS itself.

7.5. Frontend Adaptation via OODA-DOM

DOM-aware applications used OODA-DOM (Section 16) to issue headers from the browser based on local behavioral cues.

Example behaviors: - Triggering CAPTCHA dynamically: action = "challenge" - Disabling advanced UI features on high-risk sessions - Logging or sampling suspicious clickstreams for Memorize

This proves that the frontend becomes a sensor and active participant in defense, not just a passive receiver.

Each of these case studies reflects the strength of the extended OODA loop: Observe -> Orient -> Decide -> Act -> Memorize -> Adjust -> Cooperate

The next section details how OODA-HTTP wraps TLS/HTTPS to integrate seamlessly with transport-layer encryption.

8. Integration with TLS/HTTPS

OODA-HTTP is explicitly designed to operate in conjunction with existing TLS and HTTPS infrastructures. It does not modify the TLS wire format, cryptographic handshake, or encryption semantics. Instead, it introduces a behavioral and contextual overlay at the HTTP application layer, which can interact with TLS indirectly through telemetry, session metadata, and coordinated control paths.

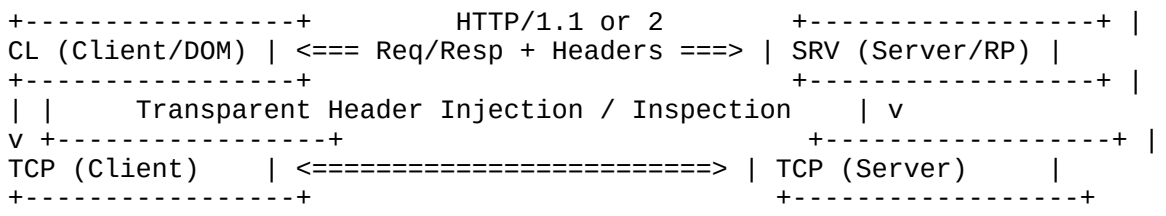
This section outlines how OODA-HTTP enhances TLS-aware systems without violating protocol separation or interoperability:

8.1. TLS Interaction Model

OODA-HTTP does not alter TLS wire formats or cryptographic protocols. Instead, it passively observes metadata exposed at the TLS termination point, such as cipher negotiation, handshake entropy, and session reuse.

8.1.0. HTTP/1.1 and HTTP/2 Interaction

OODA-HTTP is fully compatible with HTTP/1.1 and HTTP/2. It uses standard headers to carry adaptive security context across existing connections, without modifying framing or wire-level semantics.



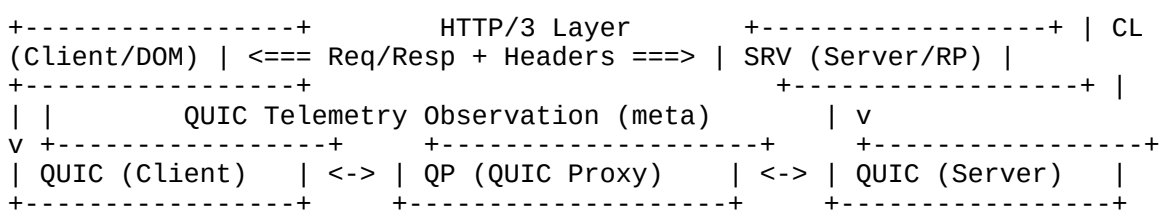
Legend: CL = Client (browser, agent) SRV = Server or reverse proxy

HTTP/1.1: Text-based protocol over TCP HTTP/2: Binary multiplexed protocol over TCP

Key properties: - OODA-HTTP headers (‘OODA-Action‘) are valid in both versions - No changes to HTTP framing or connection behavior - Compatible with intermediate proxies, CDNs, and inspection tools

8.1.1. QUIC/HTTP3 Interaction

OODA-HTTP supports QUIC by observing handshake metadata exposed at endpoints or through QUIC-aware proxies, without altering wire behavior.



Legend: CL = Client or browser (DOM-capable) SRV = Server or reverse proxy QP = QUIC proxy or observability point

QUIC = HTTP/3 transport layer over UDP

Key properties: - OODA-HTTP does not modify QUIC transport or TLS over QUIC - QUIC metadata (timing, flow IDs, handshake entropy) is passively read - Actions remain confined to the HTTP/3 layer via OODA-Action headers

8.2. Entropy Awareness and Anomaly Detection

In scenarios where entropy quality is critical - such as in IoT, embedded clients, or weak RNG environments - OODA-HTTP modules can monitor entropy characteristics of session keys or handshake fields.

Indicators of concern may include:

- Repetitive or low-entropy client random values - Predictable session resumption behavior - Use of outdated cipher suites lacking forward secrecy

Upon detection, the engine may raise the session’s threat score or flag it for deeper inspection.

8.3. Key Rotation Coordination

OODA-HTTP does not initiate TLS key exchange or renegotiation on its own. However, in deployments where TLS keying material is managed by a programmable agent (e.g., TLS termination proxy), OODA-HTTP may suggest

re-keying events in response to elevated threat scores.

This coordination is out-of-band, typically through:

- Management APIs (e.g., QUIC TLS stack integration) - IPC or shared memory hooks in NGINX/Envoy-type proxies - Dynamic reconfiguration scripts

8.4. Contextual Tagging of TLS Sessions

In multipath or multiplexed HTTP/2 and HTTP/3 environments, a single TLS connection may carry diverse request profiles. OODA-HTTP allows tagging of each request (via ‘OODA-Action‘ header) with behavioral context.

This allows:

- Session correlation for SIEM or XDR systems - Policy enforcement per HTTP request rather than per connection - Delegation of TLS-level context back to higher-layer decision engines

8.5. Post-Quantum Preparedness

While OODA-HTTP does not alter cryptographic primitives, it can be used to simulate post-quantum threat conditions:

- Triggering alerts for weak or legacy handshakes - Logging handshake entropy levels for audit or telemetry - Adapting runtime posture (e.g., throttling, challenge, isolate) based on TLS layer confidence

This bridges the gap between static TLS negotiation and real-time behavioral adaptation.

8.6. Summary

OODA-HTTP does not interfere with the TLS handshake or encryption semantics. Its role is to enrich HTTP behavior **after** the secure channel is established, by:

- Observing TLS parameters for contextual scoring - Influencing policy decisions at the HTTP layer - Supporting external coordination with TLS-aware components

In doing so, OODA-HTTP transforms passive secure channels into **context-sensitive**, **risk-aware** conduits - while remaining fully compatible with TLS 1.3, QUIC, and existing HTTPS deployments.

9. Protocol Engineering Foundations

OODA-HTTP is engineered as an overlay, not a replacement. It operates at the HTTP application layer and leverages standard-compliant header injection to carry decision signals.

The design philosophy emphasizes:

- Compatibility with all HTTP versions (1.1, 2, 3) - No changes to TCP, QUIC, or TLS wire protocols - Support for both stateless and stateful deployments - Forward extensibility through structured headers

OODA-HTTP follows key engineering principles:

9.1 Minimal Invasiveness

OODA-HTTP does not modify existing transport layers. It is deployable via proxies, middleware, service meshes, or edge nodes without changes to client libraries or core protocols.

9.2 Deterministic Behavior

Actions derived from threat scoring (e.g., block, throttle, rotate-keys) must be deterministic and auditable. Engines must avoid random behaviors that could impact traceability or compliance.

9.3 Header Formalism

The ‘OODA-Action‘ header must follow a defined schema. Optional fields may be ignored without loss of compatibility. JSON is the preferred encoding but CBOR or compact forms may be adopted in constrained environments.

9.4 Stateless by Default

Although Memorize and Adjust phases imply state, default deployments may opt for ephemeral scoring and actions without persistent memory. Stateful modes are available for advanced use cases.

9.5 Upgrade Path

OODA-HTTP supports progressive rollout. It can be introduced incrementally in reverse proxies, CDNs, or clients. Clients unaware of ‘OODA-Action‘ can still interoperate seamlessly.

9.6 Testing and Validation

Each deployment should include replay testing, entropy validation, and attack emulation scenarios. Threat simulation must include slow clients, bot agents, and low-entropy TLS sessions.

9.7 Threat Mapping Across OSI Layers

OODA-HTTP operates at the application layer but observes behaviors that span multiple OSI layers. The engine captures anomalies that originate from transport-level, session-level, or even link-level irregularities as observed through HTTP metadata.

The following attack mapping shows the relevance of each OSI layer:

Layer	Example Threats	OODA-HTTP Detection Role
7	Bot, click fraud, scraping	Behavior scoring, DOM
6	Malformed headers, encoding	Header sanitization
5	Session replay, TLS reuse	Entropy check, ID trace
4	Slowloris, burst TCP flood	Timing analysis, throttle
3	IP spoofing, routing games	Context fingerprinting
2	MAC spoofing (via proxy)	Trust anchor misalignment

Although not a transport-layer protocol, OODA-HTTP enables context-aware mitigation based on symptoms emerging at lower layers.

This aligns with STAMP-based reasoning and reinforces cooperative defense with systems like DOTS, TLS inspection proxies, or edge SIEMs.

10. Security and Privacy Considerations

OODA-HTTP introduces active decision elements into HTTP traffic. This requires careful analysis of how scores, actions, and behavioral memory are processed and shared.

10.1 Integrity of OODA-Action Headers

Intermediaries must validate the authenticity and integrity of ‘OODA-

Action‘ headers. Unsigned or tampered headers may mislead agents into inappropriate actions. Future drafts may define a signature scheme.

10.2 Replay and Tampering Risks

As OODA-Action may influence access control or TLS rotation, replayed or modified headers can pose security risks. Servers should timestamp and scope each action to a request ID or session context.

10.3 Privacy Implications

Behavioral scoring and memory storage introduce privacy exposure. Scores must not leak identifiable traits unless explicitly authorized. Systems should anonymize session traces and comply with privacy regulations.

10.4 False Positives and Negatives

Adaptive systems may misclassify benign traffic. It is critical to allow feedback loops, override mechanisms, and audit trails. Tuning must balance detection with usability.

10.5 Scope of Defense

OODA-HTTP defends at the HTTP edge. It is not a substitute for transport encryption, network ACLs, or endpoint security. It complements these by introducing application-level adaptivity.

10.6 Trust and Delegation

In multi-layer deployments (CDN, reverse proxy), action authority must be clearly defined. Misaligned trust assumptions may cause conflict between layers or over-block legitimate traffic.

11. IANA Considerations

This document registers the following HTTP header under the "Permanent Message Header Field Names" registry:

Header Field Name: OODA-Action Applicable Protocol: HTTP Status: Provisional Change Controller: IETF Specification Document: This document

This header was initially named X-OODA-Action in early drafts. Following community feedback and IETF best practices, the "X-" prefix was removed to reflect intended standardization.

Future updates may request ALPN or TLS extension identifiers as inter-protocol coordination evolves.

12. Vector Engine and Temporal Memory

OODA-HTTP integrates a semantic vector engine that captures behavioral signals and organizes them into structured vectors, enabling dynamic learning and memory-driven decisions.

12.1 Observation Vectors

Each HTTP request generates an observation vector composed of metadata, TLS negotiation traits, entropy metrics, timing signals, and navigation behavior. This vector becomes the atomic input for the semantic engine.

12.2 Orientation and Semantic Space

Observation vectors are mapped into a semantic space using anomaly detection models such as SVDD. Hyperparameter tuning is performed in real time via PSLPSO, a self-adaptive optimization technique. This

enables the system to learn new patterns incrementally.

12.3 Temporal Memory

Two memory layers are maintained:

- Short-term memory detects session bursts, replay attacks, or timing anomalies across consecutive requests.

- Long-term memory records entropy trends, vector signatures, and evolving identity traits. It enables slow-pattern detection and behavioral drift monitoring.

12.4 Learning by Activity

The vector engine evolves with traffic. Each new request contributes to pattern refinement. Memory is not static: like a human pilot, the system builds instinct through repeated exposure. This supports anticipatory security and contextual decision-making.

13. Interoperability with External Agents

OODA-HTTP is designed to interoperate with external protocols and engines to enable layered, cognitive defense. It acts as a coordination layer that synthesizes behavior, decisions, and threat intelligence across domains.

13.1 DOTS (DDoS Open Threat Signaling)

OODA-HTTP supports coordination with DOTS (RFC 9244) by:

- Receiving upstream mitigation signals from a DOTS server and converting them into contextual HTTP responses using OODA-Action headers.

- Exporting local threat scores or observations toward DOTS clients or collectors, facilitating upstream analytics or multi-tenant mitigation.

13.2 SVDD (Support Vector Data Description)

During the Orient phase, SVDD is used to learn the hyperspace of normal behavior. Each incoming request vector is checked for proximity to this boundary. Anomalous vectors increase the risk score for Decide phase logic.

13.3 PSLPSO (Polynomial Self-Learning Particle Swarm Optimization)

PSLPSO tunes the SVDD model’s hyperparameters over time. This includes:

- Radius tightening/loosening based on feedback. - Adjusting thresholds adaptively as behavior evolves. - Auto-rebalancing class boundaries with no supervision.

13.4 STAMP (System-Theoretic Accident Model and Processes)

STAMP provides a causal framework for analyzing behavioral incidents. OODA-HTTP uses STAMP integration to:

- Trace anomalous patterns to root causes (bad session chains). - Feed Memorize and Adjust phases with causal awareness. - Support forensic correlation and explainability of decisions.

13.5 Semantic Fusion

OODA-HTTP can merge these signals into a semantic decision vector. Example function:

```
orient_merge(dot_signal, svdd_score, stamp_root);
```

The result feeds Decide and Act logic, giving OODA-HTTP a predictive, interoperable edge. This forms the backbone of a layered cyber-defense architecture that evolves through feedback, learning, and cooperation.

13.6 Comparative Advantage: Application-Layer Defense

While DOTS (RFC 9244) supports upstream mitigation signals, it lacks application-layer insight. It cannot process user behavior, TLS entropy, or HTTP headers. OODA-HTTP fills this cognitive gap.

OODA-HTTP provides real-time, semantic decision-making at the edge or server, complementing DOTS at the transport/network layers.

Without OODA-HTTP, DOTS is blind to:

- Camouflaged bots or HTTP anomalies. - Low-entropy TLS handshakes. - Inconsistent navigation or session flow.

Together, they form a layered defense model:

DOTS absorbs volume. OODA-HTTP interprets meaning. They are complementary: pressure relief and adaptive reflex.

14. Strategic Doctrine and Cybersecurity Philosophy

OODA-HTTP is not only a protocol; it is a doctrine. It redefines security as a continuous decision cycle that adapts to adversaries in real time. Traditional HTTP security is reactive and passive. OODA-HTTP is proactive, evolutionary, and collaborative.

This doctrine is built on two fundamental laws:

Law #1 - Adaptive Memory Law: A system that does not remember cannot evolve. OODA-HTTP embeds memory as a first-class protocol construct, ensuring that each observation improves future decisions.

Law #2 - Cooperation over Isolation: A defense that does not cooperate is already defeated. OODA-HTTP is designed for interoperation with DOTS, TLS inspectors, STAMP, and future telemetry agents.

14.1. Key Transparency and Behavioral Fingerprinting

OODA-HTTP introduces key transparency mechanisms at the behavioral level. Rather than verifying certificates alone, it memorizes and correlates key material (e.g., TLS session fingerprints) across time, ASN, and context.

The module ‘ooda_memorize.c‘ tracks fingerprint evolution across sessions.

The module ‘ooda_cooperate.c‘ can export suspicious fingerprints to DOTS or SIEM systems.

The ‘Orient‘ phase can flag rapid or inconsistent key rotations.

The ‘Act‘ phase can trigger: OODA-Action: alert-key-anomaly

Transparency in OODA-HTTP is not purely cryptographic - it is contextual, temporal, and strategic. It complements existing PKI mechanisms such as Certificate Transparency (CT), OCSP, or CRLs by integrating behavioral indicators into the decision loop.

A self-defending protocol must remember the keys it trusts, not just accept them.

15. Layered Threat Classification (OSI vs OODA)

Traditional security frameworks classify threats by OSI layer: physical, network, transport, application. OODA-HTTP reclassifies them by cognitive phase.

This layered model enables cross-phase correlation and shows how OODA-HTTP complements or augments existing security layers.

Lvl	Example Threats	OODA Phase	OODA Response
L3	IP spoofing,DDoS	Observe,Memorize	rate-limit,alert
L4	Slowloris,SYN flood	Observe,Orient	drop,timeout
TLS	Entropy replay,downgrade	Orient,Memorize	key-rotate,reject
L7	Header anomalies,replay	Observe,Orient	challenge,drop
APP	Credential stuffing,bots	Decide,Act	captcha,ban,log
ML	Coordinated APT attack	Cooperate,Adjust	notify-DOTS,score++

This model reinforces that defense is no longer vertical (per layer), but looped and adaptive, where OODA phases interleave to correlate weak signals and respond intelligently.

OODA-HTTP is the first protocol to introduce horizontal correlation across vertical layers, enabling end-to-end cognitive security pipelines.

16. Frontend Sensors and OODA-DOM

Modern applications are no longer static. SPAs, PWAs, and WASM pages contain valuable behavioral signals.

OODA-HTTP introduces OODA-DOM: a frontend extension where the DOM (Document Object Model) acts as a live sensor of threat context.

16.1 DOM as a Radar

Frontend components can observe:

- Navigation patterns and rage clicks - Unexpected JavaScript injections or anomalies - Time-based interaction irregularities - Repeated DOM mutations from the same origin

These observations are encoded in client OODA-Action headers or POSTed to telemetry endpoints.

16.2 DOM Score and Visualization

The browser computes a DOM-specific score:

- DOM entropy - DOM event deviation - DOM injection suspicion

A local radar-style UI may freeze the screen or show a challenge.

16.3 Coordinated Frontend-Backend Loop

The DOM extends the Observe phase. When aligned with backend OODA-Action, both actors co-evolve:

- Client adjusts UI/reactivity - Server adjusts trust/session control

This loop builds full-stack cognitive defense. The UI becomes a trusted sensor.

OODA-HTTP is the first protocol where frontend activity contributes to protocol-level defense. UX and security merge into one loop.

17. OODA-DOM Good Practices Map

DOM observability depends on disciplined frontend engineering. This map defines secure best practices for DOM instrumentation.

17.1 Mapping Sensitive DOM Zones

Monitor areas such as:

- Login/authentication forms - Payment input fields - Focused elements with unusual timing - Listeners tied to user input (click, submit, input)

17.2 Local DOM Score Buffering

Buffer DOM threat scores in memory/session. Never persist to disk. Use thresholds to trigger server sync or visual feedback.

17.3 User Privacy and Transparency

Show UI hints when DOM abuse is detected (e.g., clipboard hijack). Never export DOM structure without consent or policy.

17.4 Client-Side Actions

Allowed local responses include:

- Blur sensitive fields - Trigger a CAPTCHA - Delay UI response - Block input events

17.5 Coordination with Server

If OODA-Action includes "dom-sync", enforce lockstep validation to ensure client-server agreement.

This map forms the basis of trusted DOM security. OODA-DOM is not only detection - it is secure collaboration.

18. State Memory and Client-Side Persistence

OODA-HTTP introduces adaptive memory at multiple layers. On the client, persistence helps maintain continuity of threat awareness.

18.1 Session vs Persistent Memory

- Session Memory: Stores DOM scores, behavior traces, or OODA-Action feedback for a single session. - Persistent Memory: Optionally stores known safe patterns or recurring anomalies across sessions.

Recommended scopes: - In-memory (volatile, privacy-safe) - sessionStorage (cleared on tab close) - IndexedDB (with expiration and scope)

18.2 Cookies and Namespaced Fields

OODA-specific cookies may be used, if: - Marked HttpOnly, Secure, and SameSite - Prefixed with ‘ooda_‘ (e.g., ooda_ctx, ooda_score) - Short-lived and rotated regularly

18.3 Synchronization with Server Memory

Client memory may sync with server via: - XHR beacon (DOM vector update) - WebSocket ping (score deltas) - OODA-Action header with trace summary

18.4 Privacy and Expiry

All client memory must: - Respect user privacy settings and legal rules (e.g., GDPR) - Expire unless renewed via active interaction - Never store passwords or raw keystrokes

In OODA-HTTP, memory builds trust. It is not a tool of surveillance.

19. Server-Side and Component-Based Integration

OODA-HTTP is framework-agnostic and supports modular or component-driven systems such as microservices or SSR libraries.

19.1 Modular Backend Integration

Each OODA phase maps to a module:

- ooda_observe.c: Extracts metadata and behavior - ooda_orient.c: Scores via SVDD/PSLPSO or rules - ooda_decide.c: Applies policy and selects action - ooda_act.c: Enforces (headers, delays, blocks) - ooda_memorize.c: Stores history and context - ooda_cooperate.c: Interfaces with DOTS, STAMP

Modules may run standalone or as a pipeline (e.g., on the edge).

19.2 Component-Level Embedding

For SSR or hybrid stacks (Next.js, Astro, Django):

- Add OODA middleware to routes - Annotate UI components with threat flags - Propagate scores into session context - Delay rendering on trust anomalies (DOM or TLS)

19.3 Reactive Architecture Patterns

Reactive components can:

- Parse OODA-Action headers per request - Adapt behavior (e.g., throttle, mask features) - Trigger DOM hooks for OODA-DOM sync

OODA-HTTP wraps business logic with adaptive defense, without changing core semantics across SSR/CSR/hybrid architectures.

20. OODA Engine Requirements and Incremental Models

Unlike traditional rule-based engines, the OODA engine is incremental, memory-aware, and built for adaptation - not static logic.

20.1 Minimum Requirements

An OODA-HTTP engine MUST:

- Process requests across 4+ phases (Observe -> Cooperate) - Maintain contextual state (trace, memory, scoring) - Support rule-based and score-based decisions - Accept dynamic rule updates (JSON, hot reload)

20.2 Incrementality as Doctrine

Incrementality is foundational - not optional.

An OODA engine MUST:

- Load new rules without restart - Adapt behavior to recent session context - Learn new patterns from adversary behavior

This requires:

- Appendable memory modules - Modular, versioned rules - Scores influenced by past traces

20.3 Execution vs Knowledge

The engine separates logic and knowledge:

- ooda_engine.c: core execution loop - ooda_registry.json: knowledge base - ooda_score.json: session memory - ooda_cooperate.c: inter-agent logic

This allows modular testing and external model training.

20.4 Commentary: Doctrinal Law

Rule #1: You don’t code a defense. You model the real.

A static system defends what it knows. An OODA engine defends what it learns.

The OODA-HTTP engine is not a firewall. It is a loop - adaptive, temporal, and adversary-aware.

21. Security Considerations

OODA-HTTP is designed as a security extension for HTTP. Implementations must consider the confidentiality of telemetry exchanged with the OODA engine, and the risk of false positives leading to denial of service.

22. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, May 2017.

[RFC9000] Iyengar, J. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, May 2021.

[RFC9110] Fielding, R., Nottingham, M., and J. Reschke, "HTTP Semantics", RFC 9110, June 2022.

[RFC9113] Thomson, M., et al., "HTTP/2", RFC 9113, June 2022.

[RFC9114] Bishop, M., "HTTP/3", RFC 9114, June 2022.

12.2. Informative References

[RFC9132] Mortensen, A., et al., "DDoS Open Threat Signaling (DOTS) Signal Channel Specification", RFC 9132, September 2021.

[RFC9244] Boucadair, M., et al., "Template for DOTS Signal Channel Extensions", RFC 9244, August 2022.

[TLS] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, August 2018.

[DOM] W3C, "Document Object Model (DOM)", https://www.w3.org/DOM/

[QUIC-LB] Duke, M., "Load Balancers for QUIC", draft-ietf-quic-load-balancers-21, IETF Work in Progress, July 2025.

[SIEM] Gartner IT Glossary, "Security Information and Event Management (SIEM)", 2024. https://www.gartner.com/en/information-technology/glossary/siem

[STAMP] Leveson, N., "Engineering a Safer World: Systems Thinking Applied to Safety", MIT Press, 2011.

[SVDD] Tax, D., and Duin, R., "Support Vector Data Description", Machine Learning, 54, 45-66, 2004.

[PSLPSO] Du, Z., "An Enhanced Particle Swarm Algorithm with Self-learning Strategy for Model Optimization", Springer Journal of Information Science, Vol. 60, 2023.

Appendix A. Developer Guidance

This appendix provides practical advice for developers implementing or integrating OODA-HTTP modules, headers, and logic in real-world systems.

A.1 Header Parsing

Clients and servers should parse the ‘OODA-Action‘ header as JSON. Use robust parsers with schema validation to avoid malformed input. Unknown fields MUST be ignored unless explicitly defined by policy.

A.2 Stateless vs Stateful Modes

Start with a stateless deployment to evaluate impact. Enable memory modules (e.g., Memorize, Adjust) once behavioral baselines are learned. Stateful modes require secure storage of session traces and scores.

A.3 Integration Points

OODA-HTTP can be integrated at:

- Reverse proxies (e.g., NGINX, Envoy) via middleware - Application servers via request/response filters - Frontend DOM (via OODA-DOM module) - CDN or edge layers for pre-screening traffic

Choose the point that balances latency, visibility, and control.

A.4 Debugging and Observability

Expose debug headers or logging modes for development. Example:

OODA-Debug: score=88, vector=23f..., model=svdd_v2

Track model version, thresholds, and source of decision (e.g., proxy vs origin). Use structured logs for telemetry fusion and anomaly tracing.

A.5 Fallback Mechanisms

If the engine is unavailable, default to a safe policy (e.g., allow). Ensure fallback paths are well-defined to prevent denial of service from internal errors or misconfigurations.

A.6 Rate Limiting and Cost Control

Scoring and model execution can be expensive. Use rate limiting or sampling to reduce computational load. Prioritize high-risk sessions or unusual patterns for deeper inspection.

A.7 Compatibility Tips

Ensure compatibility with:

- HTTP/1.1: plain text headers, no multiplexing - HTTP/2 and 3: multiplexed requests, header compression (HPACK/QPACK) - Intermediaries: proxies must forward or merge headers carefully

Respect header case-insensitivity and header field size limits.

A.8 Security Hardening

- Sanitize all input before scoring - Enforce strict JSON schema - Limit memory retention periods - Protect scoring models and training data

Security of the engine is critical - attackers may attempt to mislead or poison decision logic. Appendix B. Unified OODA Action Registry

This registry defines the standardized values and fields for the ‘OODA-Action‘ header. It serves as a reference for implementers to ensure interoperability across modules, vendors, and deployments.

B.1 Core Fields

Each ‘OODA-Action‘ header MUST contain:

- score (integer) - Range: 0-100. Indicates threat level. - category (string) - Classification of context. - action (string) - Recommended response. - timestamp (string) - ISO 8601 UTC time of decision.

Optional fields:

- origin (string) - Component that issued the decision. - confidence (string) - Confidence: ’low’, ’medium’, or ’high’. - reason (string) - Short tag explaining the logic path. - vector_id (string) - Reference to semantic vector used. - model (string) - Version or name of decision engine.

B.2 Standard Categories

The following categories are RECOMMENDED:

- normal - No unusual behavior. - suspicious -

Deviation from known safe behavior. - malicious - Confirmed harmful intent. - burst - High frequency or volumetric pattern. - low-rate - Slow connection or resource exhaustion. - bot-camouflage - Mimicking human patterns with intent to evade. - unknown - Unclassified or model-inconclusive.

B.3 Standard Actions

The following actions are defined:

- allow - Permit the request normally. - block - Reject request at HTTP level. - throttle - Apply rate limit or introduce latency. - challenge - Enforce CAPTCHA, MFA, or puzzle test. - isolate - Route to sandbox, honeypot, or degraded path. - log - Flag for audit without enforcement. - escalate - Send to human operator or SIEM system. - retry - Ask client to retry with modified payload. - reroute - Divert request to alternate backend.

B.4 Extensibility Notes

- New categories and actions MUST be registered or namespaced. - Use prefixes (e.g., ‘x-orgname-action‘) for private extensions. - Parsers MUST ignore unknown fields unless policy requires otherwise.

B.5 Example

OODA-Action: { "score": 93, "category": "bot-camouflage", "action": "challenge", "origin": "reverse-proxy-uk1", "confidence": "high", "timestamp": "2025-07-08T18:02:40Z", "reason": "rotating user-agent with no Referer header" }

Authors’ Addresses

Rachid Bouziane
SecRoot.io
Villa El Majd 171, Tamsna Temara
Rabat, Morocco
Email: contact@secroot.io

Expires: March 16, 2026