CRONOS IP HOLDINGS LTD. — ENGINEERING DIVISION

CronosAirShield

Phase 1 Technical Specification

CronosGuard AP Plugin: Software-Only AirSnitch Mitigation

Document	CRONOS-AIRSHIELD-SPEC-001
Version	1.0
Date	April 13, 2026
Author	Dr. Elara Voss, PhD
Classification	CONFIDENTIAL — Engineering Internal
Status	**READY FOR DEVELOPMENT**

Target: 30 days from go-decision to deployable package on OpenWrt/DD-WRT/enterprise controllers

1. SYSTEM OVERVIEW

1.1 Purpose

CronosAirShield Phase 1 (codename: CronosGuard AP Plugin) is a pure software solution that detects and disrupts all four AirSnitch attack vectors on existing Wi-Fi access points without any hardware modifications. It operates as a set of eBPF programs attached to the AP's network interfaces plus a management daemon, deployable as an OpenWrt/DD-WRT package or a Docker container on enterprise controllers.

1.2 Design Principles

- Zero hardware changes: Runs on any Linux-based AP with kernel 5.10+ and eBPF support
- Zero client changes: No client-side software, SDK, or firmware required for Phase 1
- Data-plane speed: All detection logic runs in kernel space via eBPF XDP/TC hooks; userspace is alert/management only
- Fail-open by default: If CronosAirShield crashes, normal Wi-Fi continues unaffected; configurable fail-closed for high-security environments
- Phase 2 ready: Architecture designed for seamless Lighthouse Pulse integration without re-engineering

1.3 Threat Model Addressed

The AirSnitch vulnerability family (NDSS 2026) exploits the lack of cryptographic binding between MAC addresses, encryption keys, and IP addresses across the Wi-Fi protocol stack. Phase 1 introduces cross-layer binding enforcement at the access point, catching identity mismatches that the standard Wi-Fi stack ignores.

AirSnitch Vector	Root Cause	Phase 1 Countermeasure	Detection Confidence
V1: GTK Abuse	Shared Group Temporal Key allows unicast payload injection via broadcast frames	GTK Isolation Engine (GIE) inspects broadcast-encrypted frames and drops those carrying unicast IP payloads	99%+ (deterministic pattern match)
V2: Gateway Bouncing	Gateway forwards packets based on L3 header without validating L2 source authorization	Gateway Bounce Detector (GBD) blocks packets where L2 dst = gateway AND L3 dst = another client	99%+ (deterministic pattern match)
V3: Downlink Port Stealing	Attacker spoofs victim's MAC; AP forwards	Cross-Layer Identity Binding (CLIB) validates MAC	95%+ (first spoofed frame may arrive before detection)

	victim’s traffic to attacker	against PTK session fingerprint on every frame	
V4: Uplink Port Stealing	Attacker spoofs backend MAC (gateway/DNS); captures victim’s uplink	CLIB validates infrastructure MAC bindings; alerts on any MAC not in pre-registered infrastructure set	95%+ (same race condition caveat as V3)

2. SYSTEM ARCHITECTURE

2.1 Component Overview

CronosAirShield Phase 1 consists of five components organized into two execution planes:

Component	Execution Plane	Language	Function
CLIB (Cross-Layer Identity Binding)	Kernel (eBPF TC)	C (eBPF-restricted)	Maintains MAC/PTK/IP binding table; validates every frame against bindings
GIE (GTK Isolation Engine)	Kernel (eBPF TC)	C (eBPF-restricted)	Inspects GTK-encrypted broadcast frames; drops unicast-in-broadcast injection
GBD (Gateway Bounce Detector)	Kernel (eBPF XDP)	C (eBPF-restricted)	Detects L2/L3 mismatch pattern on gateway interface; blocks bounce routing
AirShield Daemon (airshieldd)	Userspace	Rust	Manages eBPF lifecycle; processes alerts; exposes REST API; handles quarantine
AirShield Dashboard	Userspace	TypeScript/React	Web UI for monitoring, configuration, alert review, and forensic replay

2.2 Data Flow Architecture

The following describes the packet processing pipeline for a frame arriving at the AP's wireless interface:

Step 1 — Frame Arrival: 802.11 frame arrives on wlan0. The kernel's mac80211 subsystem decrypts using the appropriate key (PTK for unicast, GTK for broadcast). At this point, the kernel knows which key was used for decryption.

Step 2 — CLIB Check (TC ingress hook on wlan0): The eBPF program reads the frame's source MAC and queries the CLIB binding table (BPF_MAP_TYPE_HASH). The binding table entry contains the PTK session ID, assigned IP, first-seen timestamp, and frame counter. If the source MAC exists but the PTK session does not match (indicating MAC spoofing), the program increments the violation counter in the map, sets a flag for the alert path, and either drops the frame (fail-closed mode) or marks it for logging (fail-open mode).

Step 3 — GIE Check (TC ingress hook on wlan0): If the frame was decrypted with a GTK (broadcast key), the GIE program inspects the IP header within the payload. If the destination IP is a unicast address (not 255.255.255.255, not a multicast range, and not the broadcast address for the subnet), this is the signature of GTK abuse injection. The frame is dropped unconditionally and an alert is generated.

Step 4 — GBD Check (XDP hook on eth0/br-lan): For packets transiting the gateway interface (the bridge or ethernet uplink), the GBD program checks whether the L2 destination MAC equals the gateway's own MAC AND the L3 destination IP belongs to another client on the local subnet. This exact pattern is the gateway bounce signature. Matching packets are dropped via XDP_DROP (fastest possible path, before sk_buff allocation).

Step 5 — Alert Processing: Violation events are written to a BPF_MAP_TYPE_PERF_EVENT_ARRAY. The airshieldd daemon reads events from this ring buffer in userspace, enriches them with client metadata (hostname, DHCP lease info, connection duration), and dispatches them to configured sinks: local log, syslog, REST webhook, and the dashboard's WebSocket feed.

Step 6 — Quarantine (optional): If the policy engine determines that a client should be quarantined (configurable: automatic after N violations, or manual via dashboard), airshieldd updates the CLIB binding table to set a quarantine flag. Subsequent frames from quarantined MACs are dropped at Step 2. For enterprise deployments, airshieldd can also issue a RADIUS Change-of-Authorization (CoA) to disconnect the client at the 802.1X level.

3. MODULE SPECIFICATIONS

3.1 CLIB: Cross-Layer Identity Binding

3.1.1 Purpose

CLIB is the core detection engine. It maintains a real-time mapping between every connected client's MAC address, the encryption session used for that client, and the IP address assigned to that client. Every inbound frame is validated against this mapping. Any mismatch indicates identity spoofing.

3.1.2 Binding Table Schema

The CLIB binding table is implemented as a BPF hash map with the following structure:

```
// Key: Client MAC address (6 bytes, padded to 8)
struct clib_key {
    __u8  mac[6];
    __u16 pad;
};

// Value: Binding record
struct clib_entry {
    __u32 ptk_session_id;    // Hash of PTK material (from hostapd event)
    __u32 ip_addr;           // Assigned IPv4 (from DHCP snoop or ARP)
    __u8  ip6_addr[16];      // Assigned IPv6 (from NDP snoop)
    __u64 first_seen_ns;     // Timestamp of binding creation (ktime_get_ns)
    __u64 last_seen_ns;      // Timestamp of most recent valid frame
    __u32 frame_count;       // Total frames from this binding
    __u32 violation_count;   // Frames that failed validation
    __u8  quarantined;       // 1 = drop all frames from this MAC
    __u8  infra_device;      // 1 = pre-registered infrastructure device
    __u8  flags;             // Reserved for Phase 2 (temporal binding flags)
    __u8  pad;
};
```

3.1.3 Binding Lifecycle

Creation: A new binding is created when airshieldd receives a HOSTAPD_EVENT_STA_AUTHORIZED event (triggered after successful 4-way handshake). The daemon computes ptk_session_id = BLAKE3(BSSID || STA_MAC || ANonce || SNonce) using the handshake parameters exposed by hostapd's control interface. The IP address field is populated when the first DHCP ACK for that MAC is observed (via the DHCP snoop) or when the first ARP reply from that MAC is seen.

Validation: On every inbound frame, the eBPF program performs a map lookup by source MAC. Three outcomes are possible: (1) MAC found and ptk_session_id matches → frame passes, counters updated; (2) MAC found but ptk_session_id does NOT match → SPOOFING

DETECTED, alert generated, frame dropped or flagged per policy; (3) MAC not found → UNKNOWN DEVICE, frame dropped and alert generated (device connected without completing monitored handshake).

Expiry: Bindings are removed when airshieldd receives HOSTAPD_EVENT_STA_DEAUTHENTICATED or after a configurable idle timeout (default: 300 seconds with no frames). The daemon periodically scans the map and evicts stale entries.

Infrastructure Registration: Gateway, DNS server, DHCP server, and other infrastructure MACs are pre-registered with infra_device = 1. Any frame claiming an infrastructure MAC that does not match the registered PTK session (or that arrives without a PTK session at all, as wired devices would) triggers an immediate alert. This is how uplink port stealing is detected.

3.1.4 Performance Characteristics

Metric	Target	Measurement Method
Lookup latency	< 2 µs per frame	bpf_prog_test_run with 10K iterations; measure elapsed / iterations
Map capacity	512 entries (default); configurable to 4096	Sized for typical AP client count; enterprise controllers may increase
Memory footprint	~80 bytes per entry × 512 = ~40 KB	Plus eBPF program text (~4 KB)
CPU overhead	< 0.5% of single core at 1000 frames/sec	Measured via perf stat during sustained traffic generation
False positive rate	0% for MAC/PTK mismatch (deterministic)	PTK session ID is cryptographically derived; collision probability negligible

3.2 GIE: GTK Isolation Engine

3.2.1 Purpose

GIE detects and blocks the GTK abuse attack vector. In this attack, an adversary wraps unicast IP traffic inside GTK-encrypted broadcast frames. The victim's OS accepts these frames because most operating systems (macOS, iOS, Android, as confirmed in the NDSS paper) process higher-layer unicast content delivered in GTK-encrypted broadcast frames without questioning it.

3.2.2 Detection Logic

The GIE program attaches to the TC ingress hook on the wireless interface, after mac80211 decryption. For every frame decrypted with a GTK (identified by the key type in the skb metadata), GIE inspects the inner IP header:

```
// Pseudocode for GIE detection logic
fn gie_check(skb) -> TC_ACT {
    if skb.key_type != GTK { return TC_ACT_OK; }  // Not a broadcast frame

    let ip_hdr = parse_ip_header(skb);
    if ip_hdr.is_none() { return TC_ACT_OK; }     // Not IP (e.g., ARP broadcast =
legitimate)

    let dst_ip = ip_hdr.dst_addr;

    // Check: is destination a unicast IP?
    if !is_broadcast(dst_ip) && !is_multicast(dst_ip) &&
!is_subnet_broadcast(dst_ip) {
        // UNICAST payload inside GTK frame = GTK ABUSE DETECTED
        emit_alert(ALERT_GTK_ABUSE, skb.src_mac, dst_ip);
        return TC_ACT_SHOT;  // Drop the frame
    }

    return TC_ACT_OK;  // Legitimate broadcast
}
```

3.2.3 Legitimate Broadcast Allowlist

GIE must not interfere with legitimate broadcast/multicast traffic. The following patterns are explicitly allowed:

Protocol	Pattern	Action
ARP	EtherType 0x0806, broadcast dst	ALLOW — Standard ARP operation
DHCP	UDP src 67/68, broadcast dst IP 255.255.255.255	ALLOW — DHCP discover/offer/request/ack

mDNS	UDP dst 5353, multicast dst IP 224.0.0.251	ALLOW — Bonjour/Avahi service discovery
SSDP	UDP dst 1900, multicast dst IP 239.255.255.250	ALLOW — UPnP device discovery
IGMP	IP protocol 2, multicast dst	ALLOW — Multicast group management
IPv6 NDP	ICMPv6 types 133-137, link-local multicast	ALLOW — Router/Neighbor solicitation and advertisement
Unicast IP in GTK frame	Any unicast dst IP inside GTK-decrypted frame	DROP + ALERT — GTK abuse signature

3.2.4 Optional: Per-Client GTK Rotation

As an additional hardening measure, CronosAirShield can instruct hostapd to rotate the GTK more frequently than the default interval. This reduces the window during which a captured GTK is valid for injection.

Configuration	GTK Rotation Interval	Trade-off
Default (no change)	3600 seconds (1 hour)	Standard Wi-Fi behavior; maximum GTK abuse window
Moderate hardening	60 seconds	Manageable overhead; significantly reduces attack window
Aggressive hardening	10 seconds	Higher handshake overhead (~100 bytes per client per rotation); extremely narrow attack window
Per-client GTK (Phase 3 prep)	N/A — unique GTK per client	Eliminates GTK abuse entirely; requires significant hostapd modification; reserved for Phase 3

3.3 GBD: Gateway Bounce Detector

3.3.1 Purpose

GBD prevents the gateway bouncing attack, where an adversary sends packets to the AP/gateway that are addressed at Layer 2 to the gateway itself, but at Layer 3 to another client's IP. The gateway, seeing its own MAC as the L2 destination, accepts the frame. It then reads the L3 destination IP, finds it belongs to a local client, and forwards it—effectively allowing one client to reach another through the gateway as a relay, bypassing client isolation.

3.3.2 Detection Logic

GBD runs as an XDP program on the gateway's LAN-facing interface (typically br-lan on OpenWrt). XDP is chosen over TC because it operates before sk_buff allocation, providing the fastest possible drop path.

```
// Pseudocode for GBD detection logic
fn gbd_check(xdp_md) -> XDP_ACTION {
    let eth = parse_eth_header(xdp_md);
    let my_mac = get_gateway_mac();  // Loaded from BPF map at program init

    // Only inspect frames addressed to our MAC (standard gateway traffic)
    if eth.dst_mac != my_mac { return XDP_PASS; }

    let ip_hdr = parse_ip_header(xdp_md);
    if ip_hdr.is_none() { return XDP_PASS; }

    let dst_ip = ip_hdr.dst_addr;
    let src_ip = ip_hdr.src_addr;

    // Check: is the L3 destination a LOCAL client (not upstream/internet)?
    if is_local_subnet(dst_ip) && dst_ip != my_ip && src_ip != my_ip {
        // L2 addressed to gateway, L3 addressed to local client
        // This is the GATEWAY BOUNCE signature
        emit_alert(ALERT_GW_BOUNCE, eth.src_mac, src_ip, dst_ip);
        return XDP_DROP;
    }

    return XDP_PASS;
}
```

3.3.3 Exception Handling

Certain legitimate traffic patterns match the gateway bounce signature and must be exempted:

Exception	Pattern	Handling

Gateway-initiated traffic	src_ip == gateway_ip	EXEMPT — Gateway legitimately communicating with clients (DHCP, DNS responses)
NAT hairpin / loopback	dst_ip == gateway_ip (misrouted)	EXEMPT — Client reaching gateway-hosted services
Inter-VLAN routing	dst_ip in different subnet, routed via gateway	EXEMPT — Legitimate L3 routing (distinguished because dst is not on local wireless subnet)
Captive portal redirects	Gateway HTTP redirect to local client	EXEMPT — Added to allowlist during captive portal session

3.3.4 Performance

Metric	Target	Notes
Drop latency	< 100 ns per packet	XDP processes before sk_buff allocation; near-NIC-speed operation
Throughput impact	< 0.1% at 1 Gbps	Two map lookups (gateway MAC, local subnet) per packet; both cached in eBPF JIT
False positive rate	0% with exception list	Exception list covers all known legitimate gateway-to-client patterns

3.4 AirShield Daemon (airshieldd)

3.4.1 Purpose

airshieldd is the userspace management daemon that orchestrates all eBPF programs, processes alerts, enforces quarantine policy, and exposes the REST API for dashboard and external integrations. It is written in Rust using the Aya eBPF framework for lifecycle management.

3.4.2 Core Responsibilities

eBPF Lifecycle Management: Compiles, loads, attaches, and monitors all eBPF programs. On startup, airshieldd pins programs to /sys/fs/bpf/airshield/ for persistence across daemon restarts. It monitors eBPF verifier output and program health via bpf_prog_info queries every 10 seconds.

hostapd Integration: Connects to hostapd's Unix domain control socket (/var/run/hostapd/wlan0) to receive station events: STA_AUTHORIZED, STA_DEAUTHENTICATED, EAPOL key exchange completion. Extracts PTK-related parameters for CLIB binding table population. Also issues DEAUTHENTICATE commands for quarantined clients.

DHCP Snoop: Monitors DHCP transactions on the bridge interface to populate IP address bindings in the CLIB table. Parses DHCPACK packets to extract MAC-to-IP assignments. Falls back to ARP table monitoring if DHCP snooping is unavailable.

Alert Processing Pipeline: Reads events from the BPF perf event array ring buffer. Each event is enriched with client metadata (hostname from DHCP, connection duration, signal strength if available via nl80211), classified by severity, and dispatched to all configured output sinks.

Quarantine Engine: Implements configurable quarantine policy. Default policy: auto-quarantine after 3 violations within a 60-second window. Quarantine sets the quarantined flag in the CLIB binding table (immediate eBPF-level drop) and optionally issues a RADIUS CoA Disconnect-Request to the RADIUS server for 802.1X environments.

3.4.3 REST API

Endpoint	Method	Description	Auth
/api/v1/status	GET	System health: eBPF program status, binding table size, uptime	API key
/api/v1/bindings	GET	Current binding table with all client entries	API key
/api/v1/bindings/{mac}	GET	Single binding entry with full history	API key
/api/v1/alerts	GET	Alert log with filtering (time, type, severity, MAC)	API key

/api/v1/alerts/stream	WebSocket	Real-time alert stream for dashboard	API key
/api/v1/quarantine	GET	Currently quarantined clients	API key
/api/v1/quarantine/{mac}	POST	Manually quarantine a client	API key + admin
/api/v1/quarantine/{mac}	DELETE	Release a client from quarantine	API key + admin
/api/v1/config	GET/PUT	Read/update runtime configuration	API key + admin
/api/v1/infra	GET/POST/DELETE	Manage infrastructure device registry	API key + admin
/api/v1/metrics	GET	Prometheus-compatible metrics endpoint	None (configurable)

3.4.4 Configuration Schema

```
# /etc/airshield/airshield.toml

[general]
mode = "fail-open"           # fail-open | fail-closed
log_level = "info"           # debug | info | warn | error
binding_table_size = 512     # Max simultaneous client bindings
binding_idle_timeout = 300   # Seconds before stale binding eviction

[detection]
clib_enabled = true          # Cross-Layer Identity Binding
gie_enabled = true           # GTK Isolation Engine
gbd_enabled = true           # Gateway Bounce Detector
gtk_rotation_interval = 60   # GTK rotation period in seconds (0 = disabled)

[quarantine]
auto_quarantine = true       # Auto-quarantine on violation threshold
violation_threshold = 3      # Violations before auto-quarantine
violation_window = 60        # Window in seconds for threshold counting
quarantine_duration = 3600   # Default quarantine duration in seconds
radius_coa_enabled = false   # Send RADIUS CoA on quarantine
radius_server = ""           # RADIUS server address
radius_secret = ""           # RADIUS shared secret

[alerts]
syslog_enabled = true
syslog_facility = "local0"
webhook_enabled = false
webhook_url = ""
webhook_secret = ""

[api]
```

```
listen_addr = "127.0.0.1"   # Dashboard listen address
listen_port = 8443           # HTTPS port
api_key = ""                 # Generated on first run if empty
tls_cert = "/etc/airshield/cert.pem"
tls_key = "/etc/airshield/key.pem"

[infrastructure]
# Pre-registered infrastructure MACs
gateway_macs = ["AA:BB:CC:DD:EE:FF"]
dns_macs = []
dhcp_macs = []
```

3.5 AirShield Dashboard

3.5.1 Purpose

The AirShield Dashboard is a single-page React application served by airshieldd on the configured HTTPS port. It provides real-time visibility into network identity state, active attacks, and quarantine management.

3.5.2 Dashboard Views

View	Content	Update Frequency
Network Overview	Connected client count, active bindings, violations/minute sparkline, system health indicators (eBPF program status, CPU/memory usage)	Every 5 seconds (polling)
Binding Table	Sortable/filterable table of all current bindings showing MAC, IP, PTK session age, frame count, violation count, quarantine status; click to expand full history	Real-time via WebSocket
Attack Feed	Chronological feed of all detection events with severity color coding, expandable details showing raw frame metadata, geographic client distribution if available	Real-time via WebSocket
Quarantine Manager	List of quarantined clients with quarantine reason, duration remaining, one-click release button; manual quarantine form	Real-time via WebSocket
Configuration	Full configuration editor with validation; GTK rotation toggle; detection module enable/disable; infrastructure MAC management	On demand
Forensic Replay	Queryable log of all binding events with timeline visualization; exportable as JSON for SIEM ingestion	On demand (queries API)

4. DEPLOYMENT SPECIFICATIONS

4.1 Target Platforms

Platform	Deployment Method	Requirements	Test Priority
OpenWrt 23.x+	IPK package via opkg	Kernel 5.15+ with eBPF support; 64MB RAM; 8MB flash (for package)	PRIMARY — Largest community base
DD-WRT (latest)	IPK package (adapted)	Similar to OpenWrt; may require custom kernel config for eBPF	SECONDARY
Ubiquiti UniFi OS	Docker container on UDM/UDM-Pro	UniFi OS 3.x+; Docker support (available on Dream Machine line)	HIGH — Major prosumer/SMB market
Generic Linux AP	Debian/Ubuntu package + systemd service	Kernel 5.10+ with BTF; hostapd with control interface enabled	HIGH — Lab reference platform
Cisco IOS-XE / Meraki	Controller-level integration (Phase 1.5)	Requires partnership; eBPF not available on IOS-XE; alternative: mirror + external appliance	FUTURE — Requires Cisco partnership

4.2 Installation Procedure (OpenWrt)

```
# 1. Add CronosAirShield repository
echo 'src/gz airshield https://pkg.cronos.network/openwrt/23.x' >> \
    /etc/opkg/customfeeds.conf

# 2. Update package lists
opkg update

# 3. Install CronosAirShield
opkg install cronos-airshield

# 4. Configure (edit or accept defaults)
vi /etc/airshield/airshield.toml

# 5. Register infrastructure devices
airshield-cli infra add --mac $(cat /sys/class/net/br-lan/address) --role gateway

# 6. Enable and start
service airshieldd enable
service airshieldd start

# 7. Verify eBPF programs loaded
```

```
airshield-cli status
# Expected output:
#   CLIB:  ATTACHED to wlan0 (TC ingress)    [OK]
#   GIE:   ATTACHED to wlan0 (TC ingress)    [OK]
#   GBD:   ATTACHED to br-lan (XDP generic)  [OK]
#   Daemon: RUNNING (PID 1234, uptime 0:00:12)
#   Dashboard: https://192.168.1.1:8443
```

4.3 Hardware Test Matrix

Phase 1 will be validated against the routers specifically tested in the NDSS AirSnitch paper, plus additional high-value targets:

Device	Firmware	Vectors Confirmed Vulnerable	Test Priority
Netgear Nighthawk R8000	Stock + OpenWrt	GTK, Gateway Bounce, Port Stealing	P0 — Most popular home router in paper
TP-Link Archer AXE75	Stock + OpenWrt	GTK, Gateway Bounce, Port Stealing	P0 — Wi-Fi 6E target
D-Link DIR-3040	Stock	ALL FOUR vectors (worst-case device)	P0 — Validates against hardest target
Asus RT-AX57	Stock + Merlin	GTK, Gateway Bounce, Port Stealing	P1
Tenda RX2 Pro	Stock	GTK, Gateway Bounce, Port Stealing	P1
Ubiquiti UDM Pro	UniFi OS	GTK, Gateway Bounce (enterprise target)	P0 — Enterprise market entry
GL.iNet GL-MT6000	OpenWrt (native)	To be validated	P1 — OpenWrt reference hardware
Cisco Catalyst 9100 AP	IOS-XE (controller)	GTK, Port Stealing (from NDSS paper)	P2 — Requires controller integration

5. DEVELOPMENT PLAN

5.1 Sprint Schedule

Week	Sprint	Deliverables	Acceptance Criteria
1	Foundation	Dev environment on GL-MT6000 OpenWrt; eBPF toolchain (clang/llvm/libbpf) validated; hostapd control socket integration; CLIB binding table PoC (create/lookup/delete)	CLIB map populated on client connect; lookup returns correct entry for connected client
2	Core Detection	Full CLIB validation logic with PTK session checking; GIE broadcast inspection with allowlist; GBD XDP program on br-lan; Alert event pipeline to perf ring buffer	Reproducing AirSnitch PoC tool (from GitHub) triggers detection alerts for GTK abuse AND gateway bounce AND port stealing
3	Daemon & API	airshieldd Rust daemon with Aya eBPF management; REST API (all endpoints); quarantine engine with configurable policy; hostapd event processing; DHCP snoop for IP binding	Full API functional test suite passes; quarantine blocks client within 100ms of trigger; binding table survives daemon restart via pinned maps
4	Dashboard & Packaging	React dashboard (all 6 views); OpenWrt IPK package; Docker container for UniFi; install/config documentation; validation video against 3+ AP models from test matrix	End-to-end demo: AirSnitch attack launched, detected, attacker quarantined, visible in dashboard — all within 500ms

5.2 Team Requirements

Role	Count	Skills Required	Allocation
eBPF/Systems Engineer	1	Linux kernel eBPF (TC + XDP hooks); C; libbpf or Aya; familiarity with mac80211 and hostapd internals; previous Wi-Fi security work preferred	Full-time, 4 weeks
Rust Backend Engineer	1	Rust async (tokio); REST API design; eBPF userspace management via Aya; Unix socket IPC; basic React for dashboard	Full-time, 4 weeks
QA / Attack Simulation	0.5	Wi-Fi pentesting; ability to run AirSnitch PoC from GitHub; Scapy/aireplay-ng; packet capture analysis; video production for demo	Part-time, weeks 2–4

5.3 Technology Stack

Component	Technology	Version	Rationale

eBPF programs	C (eBPF-restricted subset)	clang 17+	Direct eBPF bytecode; maximum control over program structure and verifier compliance
eBPF management	Aya (Rust eBPF framework)	0.12+	Rust-native eBPF lifecycle management; type-safe map access; no dependency on bcc/Python
Daemon	Rust + tokio	1.77+	Memory safety critical for security daemon; async for concurrent hostapd + API + ring buffer handling
REST API	axum (Rust)	0.7+	Lightweight; built on tokio; native TLS support via rustls
Dashboard	React + TypeScript	18+	Standard SPA; minimal bundle for resource-constrained APs; WebSocket for real-time updates
Build system	Cross (Rust cross-compilation)	Latest	Cross-compile Rust for MIPS/ARM targets (OpenWrt routers)
Packaging	OpenWrt SDK + Docker	23.x SDK	Native IPK packaging for opkg; Docker for UniFi OS
Testing	AirSnitch PoC (GitHub)	Latest	Use the actual attack tool from the researchers to validate detection

6. TESTING & VALIDATION PLAN

6.1 Attack Reproduction Test Suite

Every AirSnitch vector must be reproduced in the lab and validated against CronosAirShield detection. The test harness uses the official AirSnitch PoC tool from Mathy Vanhoef's GitHub repository.

Test ID	Vector	Procedure	Expected Result	Pass Criteria
T-001	GTK Abuse	Run airsnitch.py --c2c-gtk-inject between two clients on same AP with isolation enabled	GIE drops injected frame; alert generated with type=GTK_ABUSE, source MAC, payload summary	Frame never reaches victim; alert in dashboard < 200ms
T-002	Gateway Bounce	Run airsnitch.py --c2c-ip between two clients on same AP with isolation enabled	GBD drops bounce packet at XDP layer; alert generated with type=GW_BOUNCE, source/dest IPs	Packet never forwarded; XDP_DROP confirmed via bpftool prog show counters
T-003	Downlink Port Steal	Spoof victim MAC using macchanger; send frames from attacker with victim's MAC	CLIB detects PTK session mismatch; alert generated with type=MAC_SPOOF; attacker quarantined after threshold	Alert within 100ms of first spoofed frame; quarantine within configured threshold
T-004	Uplink Port Steal	Spoof gateway MAC from attacker device; attempt to receive victim's uplink traffic	CLIB detects infrastructure MAC violation; immediate alert and quarantine	Alert generated; no traffic diverted to attacker
T-005	Cross-SSID Attack	Configure AP with main + guest SSID; attempt GTK abuse from guest to main network	GIE detects cross-SSID injection (GTK shared between SSIDs on same hardware)	Frame dropped; alert shows cross-SSID context
T-006	False Positive	Normal browsing, streaming, video calls, IoT devices for 24 hours on protected AP	Zero false positive alerts; all legitimate traffic passes without degradation	0 alerts generated; throughput within 1% of baseline

6.2 Performance Benchmarks

Benchmark	Tool	Baseline (no AirShield)	Target (with AirShield)	Max Degradation
TCP throughput	iperf3 (30s, 10 streams)	Measure per-device	Within 1% of baseline	2% hard limit

UDP throughput	iperf3 (30s, 1Mbps target)	Measure per-device	Within 1% of baseline	2% hard limit
Latency (ping)	ping -c 1000 to gateway	Measure per-device	< 0.5ms increase in mean	1ms hard limit
Jitter	iperf3 UDP jitter measurement	Measure per-device	< 0.2ms increase	0.5ms hard limit
AP CPU utilization	top / mpstat during iperf3	Measure per-device	< 2% increase	5% hard limit
AP memory usage	free during 50-client load	Measure per-device	< 5MB increase	10MB hard limit
Client connect time	Time from association to DHCP ACK	Measure per-device	< 50ms increase	200ms hard limit

7. PHASE 2 READINESS

Phase 1 architecture is explicitly designed to enable seamless Phase 2 (Lighthouse Pulse) integration. The following integration points are pre-built in Phase 1:

7.1 Pre-Built Integration Points

Phase 1 Component	Phase 2 Integration Point	What Changes
CLIB binding table	Add lighthouse_epoch field (already reserved as flags byte)	Binding creation records current Lighthouse epoch; validation checks temporal consistency
airshieldd daemon	Add Lighthouse Pulse client (WebSocket subscriber)	Daemon subscribes to ws://lighthouse.cronos.network/epochs; feeds epoch to CLIB map updates
REST API	Add /api/v1/lighthouse endpoint	Exposes current epoch, pulse health, synchronization status
Alert schema	Add temporal context to all alerts	Every alert includes the Lighthouse epoch at which the violation was detected
Dashboard	Add Lighthouse status panel	Shows pulse health, epoch counter, sync drift, temporal binding age per client
Configuration	Add [lighthouse] section to airshield.toml	Pulse source URL, failover policy, epoch validation strictness

7.2 Phase 2 Upgrade Path

The Phase 1 to Phase 2 upgrade is a software-only operation requiring no re-architecture:

- Deploy GPSDO hardware ($115) at the AP controller location or use cloud-hosted Lighthouse Pulse endpoint
- Update airshieldd to include Lighthouse Pulse WebSocket client (Rust library addition; no daemon restructuring)
- Update CLIB eBPF program to read lighthouse_epoch from an additional BPF map and include it in binding validation
- Deploy VDF client WASM module via captive portal injection for browsers or lightweight native client for IoT
- Update dashboard to show temporal binding health and Lighthouse synchronization status

Estimated Phase 2 integration time after Phase 1 is deployed: 4–6 weeks of engineering, running in parallel with Phase 1 customer deployment.

8. RISK REGISTER

Risk	Likelihood	Impact	Mitigation
eBPF verifier rejects CLIB program on older kernels	Medium	High — Blocks deployment on target hardware	Develop against oldest supported kernel (5.10); use CO-RE (Compile Once, Run Everywhere) via BTF; fallback to TC classifier without eBPF for legacy platforms
hostapd control interface not exposing PTK parameters	Medium	High — Cannot create PTK session binding	Use wpa_supplicant/hostapd patches from Mathy Vanhoef's own research tools; alternatively derive session ID from observable handshake nonces via packet capture on monitor interface
Performance impact exceeds 2% on low-end routers	Low	Medium — Limits deployment targets	Profile on weakest target hardware first (MIPS-based OpenWrt); optimize eBPF map access patterns; reduce GIE inspection to sampled mode if needed
False positive on legitimate broadcast application	Low	Medium — Drops legitimate traffic	Comprehensive allowlist testing in Week 4; configurable allowlist in GIE; fail-open mode as default ensures graceful degradation
Competitor releases similar tool before us	Low	High — Loses first-mover advantage	Accelerate: prioritize OpenWrt release in Week 3; blog post + media outreach on detection demo, not waiting for full dashboard polish
AirSnitch researchers update PoC to bypass detection	Low	Medium — Requires detection logic update	Phase 2 (temporal anchoring) makes bypass fundamentally harder; Phase 1 detection is updatable via daemon OTA without AP firmware changes

9. SUCCESS METRICS

Metric	Target	Measurement
Time to first deployable package	30 days from go-decision	Date of first validated IPK on OpenWrt
AirSnitch vectors detected	4 of 4 (100%)	All T-001 through T-005 tests passing
False positive rate	0% over 24-hour stress test	T-006 test passing
Throughput degradation	< 1% (2% hard limit)	iperf3 benchmark suite
GitHub stars (open-source CLIB)	500+ within 30 days of release	GitHub analytics
Partnership inquiries	3+ vendor conversations initiated	CRM tracking of inbound after launch
Media coverage	5+ security publications covering CronosAirShield	Press monitoring
Revenue (first 90 days)	$10K+ MRR from Professional/Enterprise tiers	Billing system

"Ship the detection. Let the industry see it work. Then sell them the physics."

END OF SPECIFICATION — CRONOS-AIRSHIELD-SPEC-001 v1.0
CONFIDENTIAL — Cronos IP Holdings Ltd. — April 2026