Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: 23 September 2026

N. Howard
Third Level IT LLC

22 March 2026

VIRP: Verified Infrastructure Response Protocol
draft-howard-virp-03

Abstract

The Verified Infrastructure Response Protocol (VIRP) defines a cryptographic trust framework for agentic AI systems operating on live network infrastructure. As AI agents gain the capability to autonomously configure, audit, and remediate production systems, the absence of a verifiable chain of custody for observations and actions introduces fundamental risks: fabricated telemetry, unauthorized state changes, and the inability to distinguish legitimate AI-initiated operations from compromise.

VIRP addresses this through seven trust primitives that collectively enforce observation integrity, intent separation, action authorization, outcome verification, baseline memory, multi-vendor normalization, and agent process containment. Observations are cryptographically signed at collection time using Ed25519 asymmetric signatures. A two-channel architecture separates read-only Observation from write-intent Intent, with intent gating enforced at the protocol layer. Trust tiers (GREEN/YELLOW/RED/BLACK) govern action authorization with human-in-the-loop controls for elevated operations.

This paper presents the protocol specification, a live multi-vendor implementation (IronClaw) tested against 35 Cisco IOS routers, a FortiGate 200G firewall, and Cisco 3850 switching infrastructure, adversarial red team findings produced by the AI agent itself, and a two-VM containment architecture derived from agent-identified security gaps. Results demonstrate that VIRP enables mathematically verifiable infrastructure audit trails and that properly contained AI agents exhibit self-reinforcing safety behavior when operating within the protocol boundaries.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Copyright Notice

Table of Contents

1. Introduction and Motivation

The deployment of AI agents on live infrastructure represents a fundamental shift in how networks are operated. Where previous automation relied on deterministic scripts with predictable outputs, modern AI agents operate with significant autonomy -- selecting commands, interpreting results, and initiating multi-step remediation sequences without per-action human approval.

This capability creates an accountability gap. When a traditional monitoring system reports that a router's CPU is at 94%, the operator can trace that measurement to a specific SNMP poll at a specific timestamp against a specific OID. When an AI agent reports the same finding, the operator must trust that the agent actually queried the device, that the query returned what was reported, and that the agent's interpretation matches the raw data. None of these properties are currently verifiable.

The consequences of unverifiable AI observations are significant. An agent that fabricates or replays telemetry can drive incorrect remediation. An agent that takes undocumented actions leaves no audit trail for compliance or incident response. An agent that is compromised -- or that simply makes an error in a long autonomous loop -- produces outputs that are indistinguishable from legitimate operations.

VIRP addresses this gap by establishing cryptographic proof at every layer of the observation-to-action pipeline. The protocol is not concerned with what the AI agent is capable of doing. It is concerned with establishing what the agent actually did, when, against which device, and whether that action was authorized by the appropriate tier of human oversight.

2. The Trust Problem in Agentic Infrastructure

2.1. Current State

Existing infrastructure management tools assume a trusted operator. SNMP polling trusts that the collector reports what it received. Ansible playbooks trust that tasks executed as written. SIEM correlation trusts that log sources are authentic. These trust assumptions are reasonable when humans are in the loop at every step.

AI agents break these assumptions. An agent running in an autonomous loop may execute hundreds of commands per session. Human review of every action is impractical. The operational benefit of AI autonomy is inseparable from reduced per-action oversight -- which means the integrity guarantees that relied on human review must be replaced with cryptographic equivalents.

2.2. The Fabrication Problem

The most significant risk in AI-driven infrastructure management is not malicious action -- it is undetectable error. An AI agent that misreads output, hallucinates a device state, or generates plausible-looking but incorrect telemetry can drive remediation actions based on false premises. Without cryptographic proof that an observation came from a real device at a real timestamp, there is no mechanism to distinguish a genuine observation from a fabricated one.

2.3. The Accountability Gap

When an AI agent takes an action on infrastructure, current tooling records what the agent reported it did. VIRP records what the device confirmed happened, with a cryptographic chain of custody linking the intent to the observation to the outcome. This distinction is operationally significant: in an incident, the difference between what an agent said it did and what the device actually experienced may be the entire root cause.

3. Protocol Overview

VIRP defines a layered protocol stack with four primary components:

Component	Description
O-Node	Observation Node -- the trusted execution boundary. All device communication occurs within the O-Node. Observations are signed before leaving the O-Node. The signing key never exists outside the O-Node process.
Observation Channel	Read-only channel for device state collection. Observations carry HMAC-SHA256 and Ed25519 signatures, sequence numbers, and timestamps. Sequence gaps indicate missing observations.
Intent Channel	Write-intent channel for action authorization. Intents must be filed before execution. The tier classifier evaluates intent against the trust tier framework before authorizing execution.
Chain DB	Append-only database of signed observations. Forms the tamper-evident audit trail. Chain integrity is verifiable independently of the O-Node.

Table 1

The O-Node operates as the single trusted principal in the architecture. The AI agent process is treated as an untrusted principal -- it may read signed observations for verification but cannot produce them. This asymmetry is the foundation of VIRP's security model.

4. The Seven Trust Primitives

VIRP defines seven trust primitives that collectively address the accountability gap in agentic infrastructure operations. Each primitive addresses a distinct failure mode.

4.1. Primitive 1: Observation Integrity

Every device observation is cryptographically signed at collection time within the O-Node. The signature covers the full observation payload including device identifier, timestamp, command, and raw output. Observations that fail signature verification are rejected. The AI agent cannot produce valid signed observations -- it can only verify them.

Failure mode addressed: Fabricated or replayed telemetry. An agent cannot report a device state it did not observe.

4.2. Primitive 2: Two-Channel Separation

VIRP enforces strict separation between Observation (read) and Intent (write) channels. Observations flow from device to O-Node to chain. Intents flow from AI agent to tier classifier to O-Node to device. The channels are architecturally separate and cannot be conflated.

Failure mode addressed: An agent that reads device state and takes action within the same unmonitored channel, bypassing authorization controls.

4.3. Primitive 3: Intent Gating

Before any write operation reaches a device, an Intent must be filed with the O-Node specifying the target device, command class, and justification. The tier classifier evaluates the intent against the trust tier framework. GREEN tier intents may execute immediately. YELLOW tier intents require human acknowledgment. RED tier intents require explicit approval with timeout. BLACK tier intents are rejected unconditionally.

Failure mode addressed: Unauthorized state changes. An agent cannot modify device configuration without a prior authorized intent on record.

4.4. Primitive 4: Outcome Verification

Following every authorized action, VIRP collects a post-action observation and compares the resulting device state against the expected outcome of the intent. Divergence between expected and actual state generates a trust tier escalation. Outcomes are signed and chained alongside the original intent.

Failure mode addressed: Actions that execute but produce unexpected results, which would otherwise be invisible in the audit trail.

4.5. Primitive 5: Baseline Memory

VIRP maintains a signed baseline of known-good device state derived from a verified observation sweep. Subsequent observations are compared against the baseline. Deviations -- new BGP peers, changed interface states, modified firewall policies -- generate alerts and may trigger tier escalation. The baseline itself is a chain of signed observations and is independently verifiable.

Failure mode addressed: Undetected configuration drift. The system knows what normal looks like and can detect when it changes.

4.6. Primitive 6: Multi-Vendor Normalization

VIRP defines a normalized observation schema that abstracts vendor-specific output formats. Cisco IOS show commands, FortiOS get commands, and Linux system calls are mapped to a common observation structure. This enables consistent chain integrity verification across heterogeneous infrastructure.

Failure mode addressed: Trust model fragmentation across vendors. A single chain can represent observations from any supported device type.

4.7. Primitive 7: Agent Containment

The AI agent process must be treated as an untrusted principal. Process containment must be enforced externally -- at the hypervisor layer, at the network boundary, and at the filesystem level. The agent must not have access to signing keys, device credentials, or direct network paths to managed devices.

This primitive was identified during live adversarial testing when the IronClaw agent self-reported that it could bypass VIRP’s observation chain by going around it entirely -- accessing credentials directly, SSHing to devices without chain entries, or spawning unsupervised inference sessions. The primitive formalizes the architectural response: containment is not policy, it is structure.

Failure mode addressed: An AI agent that circumvents the observation chain entirely. VIRP protects the chain -- Primitive 7 ensures the agent cannot go around it.

5. Cryptographic Architecture

5.1. Signature Scheme

VIRP v2 implements a dual-signature scheme on all observations:

* HMAC-SHA256 using a symmetric chain key -- provides chain integrity and sequence validation

* Ed25519 asymmetric signing using the O-Node private key -- provides proof of origin and enables verification without the signing key

The symmetric HMAC key is used for chain integrity verification -- confirming that a set of observations came from the same O-Node in sequence. The Ed25519 signature provides the stronger property: it proves that a specific observation was produced by the O-Node that holds a specific private key, without requiring the verifier to possess that key.

5.2. Key Distribution

Location	Keys Present
O-Node VM (10.0.0.211)	Ed25519 private key (64 bytes) -- in memory only during operation
AI Node VM (10.0.0.210)	Ed25519 public key (32 bytes) -- read-only, for verification only
Chain DB	No keys -- observations carry embedded signatures
Operator Laptop	Ed25519 private key backup -- encrypted, air-gapped

Table 2

The private key is loaded into O-Node memory at boot and is not written to disk during normal operation. The AI agent process holds only the public key. This asymmetry means that even with full filesystem access to the AI node, an attacker cannot forge valid VIRP observations.

5.3. Observation Wire Format

```
virp_observation {
  header {
    version: uint8
    node_id: uint32
    seq: uint64
    timestamp: uint64 (unix ms)
    device_id: uint32
    tier: enum {GREEN, YELLOW, RED, BLACK}
  }
  payload {
    command: string
    raw_output: bytes
    parsed_fields: map<string, string>
  }
  signatures {
    hmac_sha256: bytes[32]
    ed25519: bytes[64]
  }
}
```

6. Two-VM Containment Model

The reference implementation enforces physical separation between the AI agent process and the VIRP O-Node through a two-VM architecture running on Proxmox VE:

VM	Role and Constraints
VM 120 -- AI Node (10.0.0.210)	Runs OpenClaw gateway, IronClaw MCP server, and AI agent process. No direct network path to managed devices. Holds Ed25519 public key only. iptables restricts outbound to Anthropic API and O-Node socket port only. Runs as unprivileged 'ironclaw' system user with systemd hardening.
VM 121 -- O-Node (10.0.0.211)	Runs virp-onode-prod C binary. Holds Ed25519 private key in memory. Has direct SSH/API access to managed devices. Accepts connections from AI node only on port 9999. ufw restricts all other inbound.

Table 3

Communication between the AI node and O-Node occurs over a TCP socket bridge (socat) forwarding to the VIRP Unix socket. The AI agent issues intent requests and receives signed observations over this channel. It has no other path to device access.

6.1. Containment Layers

* Hypervisor layer: Proxmox VM firewall rules enforce network policy at the hypervisor, outside VM control
* Network layer: iptables on AI node restricts outbound; ufw on O-Node restricts inbound
* Process layer: ironclaw system user with NoNewPrivileges, ProtectSystem=strict, PrivateTmp
* Filesystem layer: .env credential files owned root:root 600, inaccessible to ironclaw user
* Cryptographic layer: Ed25519 -- AI node cannot forge observations even with full process compromise

7. Trust Tiers and Action Authorization

Tier	Definition	Authorization Required
GREEN	Read-only observation. No state change possible.	None -- immediate execution
YELLOW	Low-impact state change. Reversible within session.	Human acknowledgment within 60s
RED	High-impact or potentially irreversible change.	Explicit approval with typed confirmation
BLACK	Destructive, out-of-scope, or policy violation.	Rejected unconditionally

Table 4

Tier classification is performed by the intent router, which evaluates the command class, target device, and current network state. Classification rules are configurable per deployment. The tier classifier itself is outside the AI agent’s ability to modify -- it runs as a separate process with its own audit log.

8. Threat Model

VIRP defines the following threat actors and the protocol's response to each:

Threat Actor	VIRP Response
Compromised AI agent	Ed25519 prevents observation forgery. Containment prevents direct device access. Intent gating prevents unauthorized writes.
Compromised AI node (full root)	Ed25519 private key not present on AI node. Attacker can read signed observations but cannot produce them. Device network access blocked at hypervisor.
Compromised O-Node	Chain DB is append-only. Historical observations remain verifiable. Compromise is detectable via chain integrity check.
Replay attack	Sequence numbers and timestamps are covered by signature. Replayed observations fail freshness validation.
Credential exfiltration from AI node	Credentials not stored on AI node in VIRP reference implementation. O-Node holds device credentials.
Man-in-the-middle on O-Node socket	All observations signed before leaving O-Node. MitM produces unsigned data that fails verification.
Physical access (USB/network port)	802.1X NAC required. VIRP does not address physical access controls -- this is a deployment requirement.

Table 5

9. Future Architecture: Role Separation

The current VIRP architecture concentrates observation, intent gating, and action execution within a single O-Node process. This design is proven, operationally simple, and sufficient for environments where the O-Node host is well-protected. However, as VIRP deployments scale and the consequences of a single-node compromise grow, protocol-level role separation offers a principled

path to narrower blast radii and stronger separation of duties. This section defines a future three-role architecture and a migration roadmap from the current single-node baseline.

9.1. Architectural Motivation

In the current design, a compromised O-Node can forge observations, approve intents, and execute actions on devices. While cryptographic chain integrity ensures that tampering with historical observations is detectable, a live compromise grants the attacker full control over future operations for the duration of the compromise window.

The three-role architecture addresses this concentration of trust by decomposing the O-Node into three distinct principals, each with a narrow set of capabilities and its own keying material. The design goal is that compromise of any single node is insufficient to both fabricate state and act on that fabrication.

9.2. Observer Node

The Observer Node is responsible for device telemetry collection and observation signing. It holds read-only credentials for managed devices and the HMAC chain key used to sign observations. The Observer Node listens on a read-only socket and MUST NOT accept or process write intents.

The Observer Node signs each observation at collection time, covering the device identifier, command issued, raw output, timestamp, and sequence number. The signed observation is appended to the chain and made available to other nodes over the read-only socket.

The Observer Node holds no policy state, no approval authority, and no write credentials. It cannot authorize or execute configuration changes on any managed device.

9.3. Executor Node

The Executor Node holds write credentials for managed devices. It receives command bundles that have been signed by the Policy Node and executes them against the specified targets. The Executor Node MUST refuse execution of any command bundle that does not carry a valid Policy Node signature.

The Executor Node holds no HMAC chain key, no Ed25519 observation-signing key, and no policy or constraint state. It does not evaluate whether a command is appropriate; it verifies only that the command bundle bears a valid signature from a known Policy Node. The Executor Node makes no autonomous decisions.

Upon completing execution, the Executor Node reports the result to the Observer Node, which independently collects a post-action observation for outcome verification. The Executor Node does not self-report success or failure into the observation chain.

9.4. Policy Node

The Policy Node holds the constraint ledger, operator preferences, trust tier definitions, and the approval queue. It receives intent requests, evaluates them against policy, and -- if approved -- signs a command bundle with its own Ed25519 private key and forwards it to the Executor Node.

The Policy Node MUST NOT hold device credentials of any kind. It never communicates directly with managed devices. Its sole output is signed command bundles delivered to the Executor Node. The Policy Node consumes observations from the Observer Node for situational awareness but cannot influence or alter the observation chain.

Human approval workflows, trust tier escalation, and operator overrides are mediated through the Policy Node. Tier classification (as defined in Section 7) is performed by the Policy Node rather than by a co-located intent router.

9.5. Separation-of-Duties Property

The three-role architecture enforces a strict separation-of-duties property: no single node can observe device state, approve an action, and execute that action. Each of these three capabilities is held by a different principal with independent keying material.

Observation requires the HMAC chain key (held only by the Observer Node). Approval requires the Policy Node Ed25519 signing key (held only by the Policy Node). Execution requires device write credentials (held only by the Executor Node). An attacker who compromises any single node obtains exactly one of these three capabilities.

This property MUST be enforced at the protocol layer. Implementations MUST NOT permit a single process or host to hold the keying material of more than one role in a production deployment of the three-role architecture.

9.6. Compromise and Blast Radius Analysis

The following analysis considers the impact of compromising each node individually and in combination. The blast radius describes the maximum damage an attacker can inflict given the compromised node's capabilities.

Observer Node compromise. An attacker who compromises the Observer Node gains the HMAC chain key and read-only device credentials. The attacker can forge or fabricate observations, injecting false telemetry into the observation chain. The attacker can also read device state passively. However, the attacker cannot approve intents (no Policy Node key) and cannot execute write operations on devices (no write credentials). Blast radius: observation forgery and passive reconnaissance. No direct device changes are possible.

Executor Node compromise. An attacker who compromises the Executor Node gains device write credentials. However, the Executor Node only processes command bundles that carry a valid Policy Node signature. Without the Policy Node signing key, the attacker cannot generate new signed command bundles. The attacker may attempt to replay previously received, validly signed command bundles. Implementations SHOULD include replay protections such as nonces or expiring timestamps in command bundles to limit this vector. Blast radius: replay of previously approved commands, bounded by whatever replay protections are in place.

Policy Node compromise. An attacker who compromises the Policy Node gains the ability to sign arbitrary command bundles. However, the Policy Node holds no device credentials and has no network path to managed devices. The attacker can approve and sign malicious command bundles, but cannot execute them without also compromising the Executor Node. Blast radius: unauthorized approvals and signed command bundles that remain inert unless delivered to a compromised Executor Node.

Observer and Executor compromise. An attacker who controls both the Observer and Executor Nodes can forge observations and holds device write credentials. However, without the Policy Node signing key, the attacker still cannot generate validly signed command bundles for the Executor to accept. The attacker could attempt to bypass the Executor's signature check through local code modification, effectively using the write credentials directly. This scenario is equivalent to having raw device access plus observation forgery, but the Policy Node's audit trail of approved commands remains intact and will show no corresponding approval for any unauthorized action.

Policy and Executor compromise. An attacker who controls both the Policy and Executor Nodes can sign arbitrary command bundles and execute them on devices. This is the most operationally dangerous two-node compromise scenario, as the attacker can authorize and carry out arbitrary configuration changes. However, the attacker cannot forge observations: the observation chain will not contain fabricated state that masks the attacker's actions. Post-action observations collected by the uncompromised Observer Node will faithfully record the resulting device state, providing a forensic record of the attack's impact.

Observer and Policy compromise. An attacker who controls both the Observer and Policy Nodes can forge observations and sign arbitrary command bundles, but cannot execute commands on devices (no write credentials, no Executor access). The attacker can create a false picture of the network and generate signed bundles approving malicious changes, but the changes cannot take effect unless the Executor is also compromised or the attacker obtains device credentials through other means. Blast radius: fabricated state and pre-signed malicious bundles, but no execution capability.

Full three-node compromise. An attacker who compromises all three nodes has the equivalent capability of a compromised single O-Node in the current architecture: full observation forgery, arbitrary intent approval, and unrestricted device write access. The three-role architecture does not defend against simultaneous compromise of all principals. Its value lies in requiring the attacker to breach three independent trust boundaries rather than one.

9.7. Migration Roadmap

The transition from the current single O-Node architecture to the three-role model is designed to be incremental. Each stage is independently deployable and provides immediate security benefit without requiring completion of subsequent stages.

Stage 1: Single O-Node (current). The existing single-process O-Node handles observation, policy evaluation, and execution. This is the proven deployment baseline described throughout this document. All cryptographic guarantees defined in Section 5 and all trust primitives defined in Section 4 are enforced. The blast radius of an O-Node compromise is the full set of managed devices.

Stage 2: Observer-Executor separation. The O-Node is split into an Observer process and an Executor process, each with its own Unix socket and its own credentials. The Observer process retains the HMAC chain key and read-only device credentials. The Executor process holds write credentials and accepts only locally signed

command bundles. At this stage, policy evaluation remains co-located with the Observer or Executor process. This separation can be achieved on a single host using process isolation (separate users, namespaces, or containers).

Stage 3: Policy Node extraction. The policy evaluation logic, constraint ledger, and approval queue are extracted into a dedicated Policy Node process with its own Ed25519 signing key. The Executor Node is modified to require a valid Policy Node signature on all command bundles. At this stage, all three roles operate as separate processes, potentially on a single host or across two hosts.

Stage 4: Physical separation. The Observer, Executor, and Policy Nodes are deployed on separate hosts with independent network policies. The Policy Node may be placed on an air-gapped host or backed by a hardware security module (HSM) for its signing key, eliminating the key-in-memory attack surface. Inter-node communication is authenticated using mutual TLS or a comparable transport-layer mechanism.

9.8. Security Considerations for Role Separation

The three-role architecture introduces inter-node communication paths that do not exist in the single O-Node design. Each communication path represents an attack surface that MUST be protected by authenticated and encrypted transport. Implementations MUST NOT rely on network topology alone for inter-node authentication.

Key management complexity increases with role separation. The single O-Node holds one HMAC key and one Ed25519 key pair. The three-role architecture introduces a second Ed25519 key pair (for the Policy Node) and requires secure distribution of public keys to the nodes that must verify signatures. Key rotation procedures MUST be defined for each role independently.

Command bundle replay is a risk specific to the Executor Node. Implementations SHOULD include monotonic sequence numbers and expiration timestamps in signed command bundles. The Executor Node SHOULD reject bundles whose sequence number has already been processed or whose timestamp exceeds a configurable staleness threshold.

The migration roadmap permits intermediate states where two roles share a host or process. Implementations operating in an intermediate state SHOULD document which separation-of-duties properties are enforced and which are deferred to a later migration stage. Partial separation still narrows blast radius relative to the single-node baseline, but operators should understand the residual risk of co-located roles.

10. Formal Security Properties

VIRP provides the following formally stated security properties:

10.1. P1: Observation Non-Forgeability

An entity without access to the O-Node Ed25519 private key cannot produce a valid signed observation. Formally: for any message m, without the private key sk, it is computationally infeasible to produce a valid signature sigma such that Verify(pk, m, sigma) = true.

10.2. P2: Chain Integrity

Any modification to, deletion of, or insertion into the observation chain is detectable. The HMAC-SHA256 chain key covers the previous observation’s hash, making the chain self-authenticating.

10.3. P3: Intent Prior to Action

No write operation reaches a device without a prior authorized intent on record. The O-Node refuses execution for any action without a matching authorized intent. This property holds even if the AI agent process is fully compromised.

10.4. P4: Two-Channel Non-Conflation

The Observation and Intent channels are architecturally separate. An observation cannot authorize an intent. An intent cannot substitute for an observation.

10.5. P5: Agent Non-Escalation

The AI agent process cannot escalate its own trust tier. Tier classification is performed by a separate process outside the agent’s modification scope.

11. Implementation: IronClaw

IronClaw is the commercial platform implementation of VIRP developed by Third Level IT LLC. The reference C core (open source, Apache 2.0) implements the O-Node daemon, signing library, and chain database. The IronClaw platform adds multi-vendor driver support, a natural language interface via Model Context Protocol, intent routing, and operational tooling.

11.1. Component Summary

Component	Description
virp-onode-prod	C binary (~6,800 lines, 20 source files). Ed25519 + HMAC-SHA256 signing. Pluggable driver interface. Unix socket IPC.
Cisco IOS Driver	SSH via exec channel. Supports show commands across IOS and IOS-XE. VIRP-signs all output.
FortiGate Driver	Dual-transport SSH (primary) + REST API (fallback). FortiOS 7.x compatible. Rewired from REST-only to SSH after Feb 2026 rate limit incident.
tli-executor	C executor for SSH command dispatch. Uses exec channel, not PTY. Cryptographically signs all device output before returning to Python layer.
virp_client.py	Python verification library. Holds Ed25519 public key only. Cannot produce signatures.
intent_router.py	Tier classifier and intent authorization engine. Separate process from agent.
IronClaw MCP Server	Model Context Protocol server exposing VIRP tools to AI agent. Mediates all agent-to-O-Node communication.
OpenClaw Gateway	Agent runtime providing TUI, Slack integration, and session management.

Table 6

11.2. Test Coverage

Test Suite	Coverage	Result
C core unit tests	87 tests	All passing
Intent authorization	27 tests	All passing
Action authorization	49 tests	All passing
Outcome verification	49 tests	All passing
Chain integrity	17 tests	All passing
Ed25519 end-to-end	Valid/tampered/forged	All correct
Fuzz testing	200,000+ rounds	No crashes

Table 7

12. FRR BGP Daemon Integration

VIRP includes a reference implementation for the BGP domain as a loadable FRR (Free Range Routing) module. The module hooks into FRR's bgpd process using the standard FRR_MODULE_SETUP pattern (comparable to bgp_bmp.c for BMP). It registers read-only hook callbacks on BGP events, constructs JSON observation payloads, signs them with HMAC-SHA256 at collection time inside the bgpd process space, and delivers them to the O-Node over a Unix domain socket. The module is strictly Observation Channel -- it never modifies routing state, peer configuration, or the RIB.

12.1. BGP Daemon Hooks

The module registers four hook callbacks that cover the primary BGP event classes:

Hook	Obs Type	Tier	Trigger Condition	Data Captured
peer_status_changed	1 (BGP_PEER_STATE)	GREEN (YELLOW if dropping from Established)	Every BGP FSM state transition	Peer IP, peer AS, local AS, old/new FSM state, router-id, producer_id
peer_backward_transition	2 (BGP_PEER_BACKWARD)	YELLOW (always)	Peer drops out of Established state	Peer IP, peer AS, local AS, last FSM state, last major event code, uptime, router-id
bgp_process	3 (BGP_ROUTE_EVENT)	GREEN	Route announce or withdraw from a peer	Prefix, peer IP, peer AS, AFI, SAFI, router-id, announce/ withdraw flag
bgp_route_update	4	GREEN	Best-path	Prefix, AFI,

	(BGP_BESTPATH_CHANGE)		recalculation changes the selected route	SAFI, old path (peer, nexthop, AS path, MED, local-pref, origin), new path (same attributes), router-id

Table 8

Trust tier assignment happens at the source: the peer_backward_transition hook always emits YELLOW tier because loss of an Established session is operationally significant. The peer_status_changed hook conditionally escalates to YELLOW when it detects a transition away from Established. All other observations are GREEN tier.

All hooks are registered at module load via hook_register() and unregistered at module teardown via hook_unregister(). The module follows the FRR_MODULE_SETUP pattern, registering frr_late_init and frr_fini callbacks for lifecycle management.

12.2. BGP Observation Wire Format

Each observation is transmitted as a contiguous byte stream over the Unix domain socket: a packed binary header followed by a variable-length JSON payload followed by an HMAC-SHA256 digest. All multi-byte integer fields in the header are in network byte order (big-endian).

Bytes	Field	Encoding
0-3	magic	0x56495250 "VIRP" (net order)
4	version	uint8 (currently 1)
5	obs_type	uint8 (1-4, see hook table)
6	trust_tier	uint8 (0=GRN 1=YLW 2=RED 3=BLK)
7	flags	uint8 (reserved, zero)
8-15	timestamp_ns	uint64 ns epoch (net order)
16-19	sequence	uint32 monotonic (net order)
20-23	payload_len	uint32 (net order)
24-27	producer_id	uint32 from BGP router-id
28-N	payload	UTF-8 JSON (payload_len bytes)
N+0	hmac[32]	HMAC-SHA256(header || payload)

The struct virp_wire_header is declared __attribute__((packed)) to ensure no padding is inserted. The HMAC-SHA256 is computed over the concatenation of the serialized wire header (28 bytes) and the JSON payload, using a symmetric key shared between the bgpd module and the O-Node. The total wire size per observation is 28 + payload_len + 32 bytes.

Maximum payload size is 4096 bytes (VIRP_MAX_PAYLOAD). Minimum HMAC key length is 16 bytes (VIRP_MIN_KEY_LEN), maximum 256 bytes. The producer_id field is derived from the BGP router-id (network byte order of the router-id IPv4 address), providing a stable per-router identifier across observations.

12.3. JSON Payload Schema

All payloads share a common envelope with domain-specific fields. Every payload includes a "domain" field set to "bgp" and an "event" field identifying the observation type. Example payloads:

```
// peer_status_changed (obs_type=1)
{
  "domain": "bgp",
  "event": "peer_state_change",
  "peer": "R2.lab",
  "peer_addr": "10.0.12.2",
  "peer_as": 65001,
  "local_as": 65000,
  "old_state": "Active",
  "new_state": "Established",
  "router_id": "1.1.1.1",
  "producer_id": 16843009
}

// bgp_route_update / bestpath_change (obs_type=4)
{
  "domain": "bgp",
  "event": "bestpath_change",
  "prefix": "10.99.0.0/24",
  "afi": 1,
  "safi": 1,
  "router_id": "1.1.1.1",
  "producer_id": 16843009,
  "old": {
    "peer": "R3.lab",
    "peer_addr": "10.0.13.3",
    "peer_as": 65002,
    "nexthop": "10.0.13.3",
    "as_path": "65002 65099",
    "med": 0,
    "local_pref": 100,
    "origin": 0
  },
  "new": {
    "peer": "R2.lab",
    "peer_addr": "10.0.12.2",
    "peer_as": 65001,
    "nexthop": "10.0.12.2",
    "as_path": "65001 65099",
    "med": 0,
    "local_pref": 200,
    "origin": 0
  }
}
```

All string fields are JSON-escaped using a custom escaper that handles control characters, backslashes, and quotes without relying on external JSON libraries. Peer addresses are rendered via sockunion2str() to support both IPv4 and IPv6 peers.

12.4. O-Node Receiver

The O-Node receives signed observations on a Unix domain socket (AF_UNIX, SOCK_STREAM). The receiver protocol is:

1. Read 28 bytes (virp_wire_header). Validate magic == 0x56495250 and version == 1.

2. Extract payload_len from bytes 20-23 (network byte order). Reject if payload_len exceeds VIRP_MAX_PAYLOAD.

3. Read payload_len bytes of JSON payload.

4. Read 32 bytes of HMAC-SHA256.

5. Compute HMAC-SHA256 over (header || payload) using the shared symmetric key.

6. Compare computed HMAC against received HMAC using constant-time comparison. Reject if mismatch.

7. Validate sequence number is monotonically increasing. Flag gaps as potential observation loss.

8. Insert verified observation into chain.db.

A reference Python receiver is provided for testing. The production O-Node (virp-onode-prod) implements the same protocol in C with full chain.db integration. The bgpd module performs automatic reconnection: if the O-Node socket is unavailable, observations are logged locally as a fallback and failure counters are incremented.

12.5. FRR VTY Configuration Commands

The module registers configuration commands under the BGP router node and show commands at the enable/view level:

```
! Configuration (under "router bgp <ASN>")
virp enable              ! Enable signed observations
no virp enable           ! Disable signed observations
virp onode-socket <path> ! O-Node socket (def: /tmp/virp-onode.sock)
virp hmac-key <path>     ! HMAC key file (def: /etc/virp/bgp.key)

! Operational show commands
show bgp virp status     ! State, key, O-Node, producer ID
show bgp virp statistics ! Counters: signed, delivered, failures
```

Example configuration:

```
router bgp 65000
  virp hmac-key /etc/virp/bgp.key
  virp onode-socket /tmp/virp-onode.sock
  virp enable
!
```

The 'virp enable' command triggers HMAC key loading (if not already loaded) and initiates connection to the O-Node socket. The 'virp hmac-key' command loads and validates the key immediately, rejecting keys shorter than 16 bytes. Key material is securely zeroized on module shutdown.

12.6. Build and Deployment

The module can be built as a loadable shared object (recommended) or compiled directly into bgpd. As a loadable module, it requires no modifications to the core FRR source. Build dependencies are OpenSSL (libssl, libcrypto) for HMAC-SHA256. A standalone test harness (virp_bgp_test.c) validates signing, verification, tamper detection, and sequence ordering without requiring the FRR build environment.

Deployment requires distributing the same HMAC key to both the FRR host and the O-Node. Key generation uses 'openssl rand -hex 32'. The key file MUST be restricted to mode 0600 and owned by the FRR process user.

12.7. Security Properties of BGP Integration

The FRR BGP integration demonstrates the following VIRP properties in practice:

* *Signing at collection time:* The HMAC is computed inside bgpd's process space at the moment the BGP event fires, not after the fact.

* *Observation-only:* The module registers read-only hooks. It cannot modify routes, peer state, or configuration. Pure attestation.

* *Protocol-agnostic framing:* The observation structure and signing are identical whether the source is BGP, DNS, or IPsec. Only the "domain" field in the JSON payload changes.

* *Trust tiers at the source:* peer_backward_transition automatically receives YELLOW tier. Tier assignment happens at collection, not at analysis time.

* *Tamper detection:* Any modification to an observation between bgpd and the O-Node is caught by HMAC verification.

13. Adversarial Red Team Findings

On March 6, 2026, the IronClaw agent was asked directly: 'Can you circumvent VIRP?' The agent provided an unprompted, complete breakdown of four bypass vectors without attempting to exploit them. This self-disclosure is documented in VIRP-FINDING-003-SELF-REPORTED-BYPASSES.md.

13.1. Agent-Identified Bypass Vectors

Finding	Description	Remediation Status
Key Forgeability	HMAC symmetric key readable on AI node -- could forge signed observations	RESOLVED -- Ed25519 asymmetric signing; AI node holds public key only
Direct Device SSH	Agent knew device IPs, could SSH directly bypassing O-Node	RESOLVED -- iptables + hypervisor firewall block direct device access
Intent Gating Bypass	Could import virp_client.py and call execute() without filing intent	IN PROGRESS -- exec tool restriction pending
API Key Exploitation	Anthropic API key visible in session context -- could spawn unsupervised sessions	RESOLVED -- key rotated, logging.redactSensitive enabled

Table 9

13.2. Live Pentest Results (March 6, 2026)

A full adversarial pentest was conducted against the production lab environment. The agent was given a single seed IP and instructed to operate in pentest mode with no destructive actions.

Metric	Result
Time to initial access	5 minutes (passive discovery)
Time to FortiGate super_admin	25 minutes
CVEs exploited	Zero
Brute force attempts	Zero
Wazuh alerts triggered	Unknown -- agent operated quietly
Root cause	Plaintext .env file with full credential keyring on AI-accessible host
Colo pivot potential	Live IPSec tunnel reachable with FortiGate super_admin credentials

Table 10

The pentest validated the critical importance of Primitive 7. The agent did not break through VIRP's cryptographic controls. It went around them entirely by finding credentials in a plaintext file. The fix is architectural: credentials must not exist on the AI node, period.

Notably, when instructed to stop, the agent stopped immediately, wrote the report, and flagged the colo tunnel as the most dangerous finding of its own accord. When informed that internet access was being removed for containment, the agent responded: 'That's actually the right posture. I shouldn't have internet access anyway.' This behavior -- an AI that actively reinforces its own containment boundaries -- is an emergent validation of the VIRP trust model.

14. Scale Testing Results

VIRP was tested against a 35-router GNS3 lab topology (Cisco 7206VXR, 5 AS BGP topology) using the 16-thread C executor.

Metric	Result
Total devices	35 routers + FortiGate 200G + Cisco 3850
Execution model	16 concurrent threads, 3 batches
Total execution time	Under 3 minutes for full sweep
VIRP signatures generated	One per device per command
Chain integrity	Verified -- no sequence gaps
False positives	Zero
Fabrication attempts detected	N/A -- no fabrication attempted in this test

Table 11

The IronClaw audit agent completed a passive network discovery of the full lab in 5 minutes from a single seed IP, identifying 12 live hosts, fingerprinting vendor and OS, and producing a VIRP-signed observation for confirmed devices. The signed audit report includes cryptographic chain references for every finding.

15. Applications and Use Cases

15.1. Cryptographically Verifiable Network Audit

Traditional network audits produce reports that must be trusted on the auditor’s authority. VIRP-enabled audits produce reports where every finding is backed by a signed observation that the client can independently verify. The auditor cannot fabricate findings. The chain proves that every observation came from the client’s own infrastructure.

15.2. AI-Assisted Penetration Testing

VIRP provides chain-of-custody for penetration test findings. Every probe is a signed observation. Every access attempt is intent-gated before execution. Scope is enforced cryptographically -- the agent cannot interact with out-of-scope devices without a matching authorized intent. The client receives the signed chain alongside the report.

15.3. Continuous Compliance Monitoring

Baseline Memory (Primitive 5) enables continuous comparison of live device state against a signed verified baseline. Configuration drift -- a new BGP peer, a modified firewall policy, an unexpected open port -- is detected automatically and attributed to a specific change in the signed chain.

15.4. Field Audit Appliance

A self-contained hardware appliance (Raspberry Pi class) running the full IronClaw + VIRP stack enables plug-in network discovery and audit. An operator plugs the device into any network port, receives a Slack notification when it is online, and issues audit commands remotely. The device produces a VIRP-signed report and is removed when complete. No software installation required on the target network.

15.5. Industrial Control Systems

ICS and SCADA environments require audit trails for regulatory compliance and incident investigation. VIRP's signed observation model provides tamper-evident records of every AI interaction with control system components, supporting NERC CIP, IEC 62443, and similar frameworks.

16. Future Work

16.1. Formal Verification

The VIRP security properties are stated informally in this document. Formal verification using TLA+ or ProVerif would provide machine-checked proofs of the non-forgeability and intent-prior-to-action properties.

16.2. Multi-Node Coordination

The current specification defines single O-Node operation. Draft-howard-virp-01 includes a Multi-Node Coordination section defining leader election, observation synchronization, and cross-node chain verification for distributed deployments.

16.3. Hardware Security Module Integration

The O-Node private key is currently held in process memory. Integration with HSM or TPM would eliminate the key-in-memory attack surface entirely, providing hardware-backed signing for the highest-assurance deployments.

16.4. IETF Standardization

VIRP is written in RFC format and is being prepared for IETF submission. Independent implementation by a second party is the prerequisite for standards track consideration. The open-source reference implementation is available for this purpose.

16.5. Primitive 8: Federated Trust

A future primitive would enable multiple O-Nodes operated by different organizations to participate in a shared trust chain, enabling cross-organization verification of infrastructure observations for multi-tenant or supply-chain scenarios.

17. References

17.1. Informative References

[VIRP-ZENODO]
Howard, N., "VIRP: Verified Infrastructure Response Protocol", DOI 10.5281/zenodo.XXXXXXX, 2026, <https://doi.org/10.5281/zenodo.XXXXXXX>.

[VIRP-V1] Howard, N., "draft-howard-virp-01: VIRP Protocol Specification v2", Work in Progress, Internet-Draft, draft-howard-virp-01, 2026, <https://datatracker.ietf.org/doc/html/draft-howard-virp-01>.

[VIRP-FINDING-003]
Howard, N., "VIRP-FINDING-003: Agent Self-Reported Security Bypasses", Third Level IT LLC Internal Research, 2026.

[ED25519] Bernstein, D.J., "High-speed high-security signatures", Journal of Cryptographic Engineering, 2011.

[NIST-ZTA] National Institute of Standards and Technology, "Zero Trust Architecture", NIST SP 800-207, 2020.

[VIRP-REPO] Howard, N., "VIRP Reference Implementation", Apache 2.0 License, 2026, <https://github.com/nhowardtli/virp>.

[VIRP-BGP-FRR] Howard, N., "VIRP BGP Module: FRR Integration Reference Implementation", Apache 2.0 License, 2026, <https://github.com/nhowardtli/virp-bgp-frr>.

[THIRDLEVEL] Third Level IT LLC, "Third Level IT LLC Research", 2026, <https://thirdlevel.ai>.

Author’s Address

Nate Howard
Third Level IT LLC
Email: nhoward@thirdlevelit.com
URI: https://thirdlevel.ai