SEAS-8414 CYBER ANALYTICS

# 03 Protocol Security Engine

Grammar Inference, Stateful Fuzzing & Zero-Day Discovery

Dr. Mallarapu · Breakwater Security Platform

# Phase 2 Recap: Firmware Intelligence

## Eight sprints from extraction to compliance

Sprint 1
**Firmware Intel**
Extraction & parsing

Sprint 2
**SBOM Generation**
Component inventory

Sprint 3
**Component CVE**
Library vulnerability

Sprint 4
**Binary Analysis**
Static disassembly

Sprint 5
**Fuzz Testing**
Input mutation

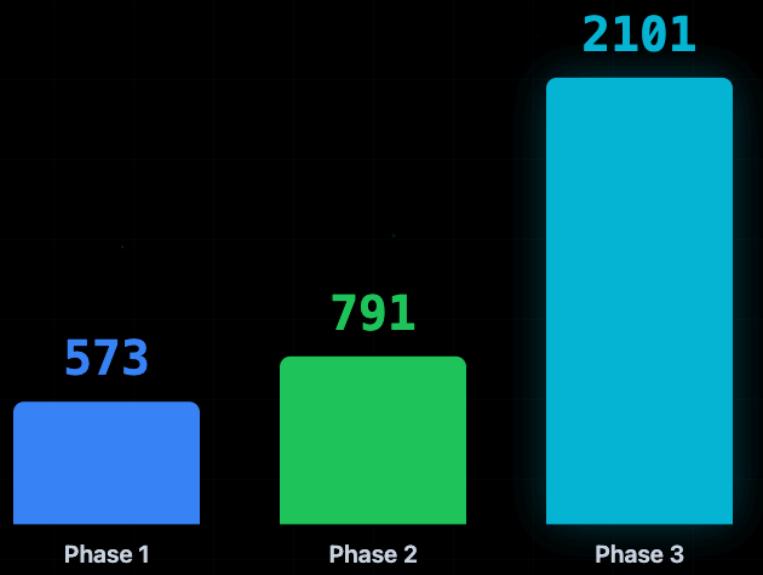
Sprint 6
**LLM Taint**
AI-guided taint flow

Sprint 7
**Integrity Check**
Hash verification

Sprint 8
**Compliance**
Regulatory mapping

# Test Progression

Growing confidence with every phase




3.7x test growth from Phase 1 to Phase 3

# The Research Question

## Why does protocol security deserve its own phase?

- Protocols ARE the attack surface
- MITRE ATT&CK ICS: T0886, T0866, T0862
- Behavioral assessment, not configuration check

SECTION 02

# The Phase 2 Limitation

What firmware analysis cannot tell us

# Static vs. Protocol Analysis

## Two complementary lenses

### Static Analysis

- Examines code in isolation
- Finds strcpy, sprintf bugs
- Binary-level view

VS

### Protocol Analysis

- Examines code in context
- Tests live network interaction
- Runtime behavior view

Phase 2 = static · Phase 3 = runtime

# Case Study: CVE-2021-36260

## Hikvision command injection via HTTP


**9.8** CVSS Critical
CVE-2021-36260

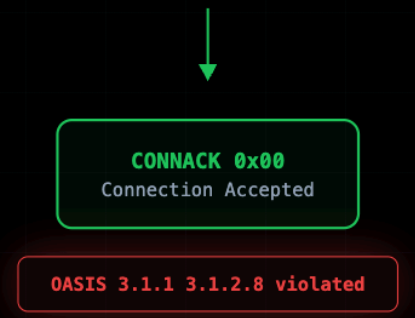
70+ camera models affected worldwide

# MQTT Authentication Bypass

## CONNECT without credentials accepted

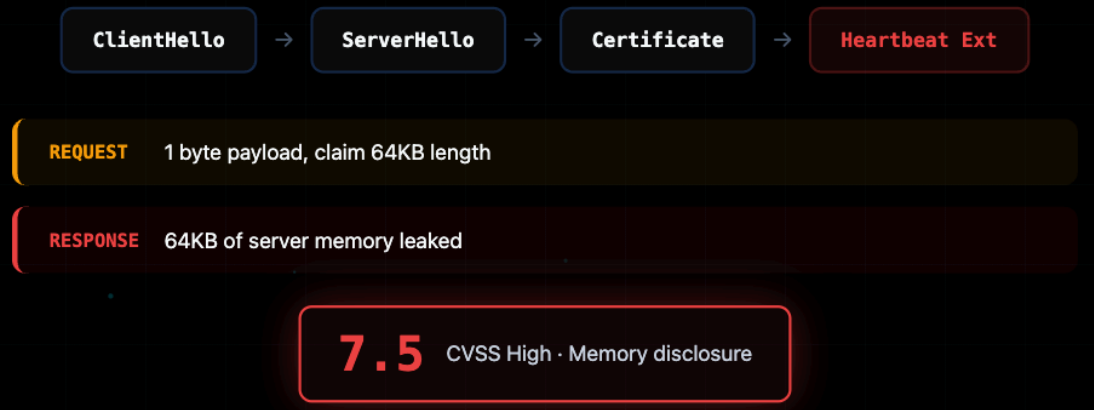
CONNECT PACKET

| | | |
|---|---|---|
| Fixed Header | 0x10 | |
| Protocol Name | MQTT | |
| Protocol Level | 0x04 | |
| Connect Flags | 0x00 | No username/password |
| Client ID | attacker-1 | |



# Heartbleed: A Protocol-Level Bug

## CVE-2014-0160 in TLS heartbeat extension



# The Critical Gap

**The gap between potential behavior and actual behavior is where the most dangerous vulnerabilities live.**

**STATIC**
What code CAN do

VS

**PROTOCOL**
What device ACTUALLY does

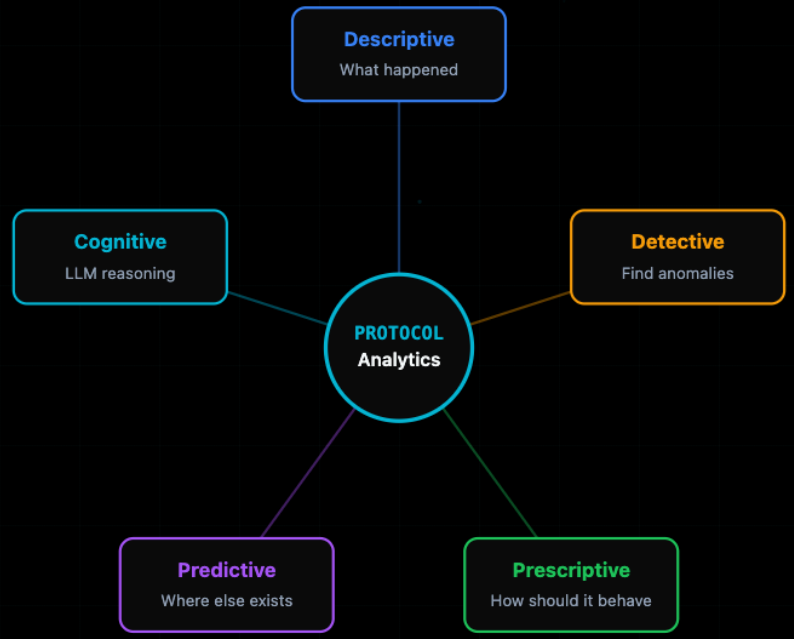
SECTION 03

# Phase 3: Twelve Sprints

Protocol security from transcripts to exploit chains

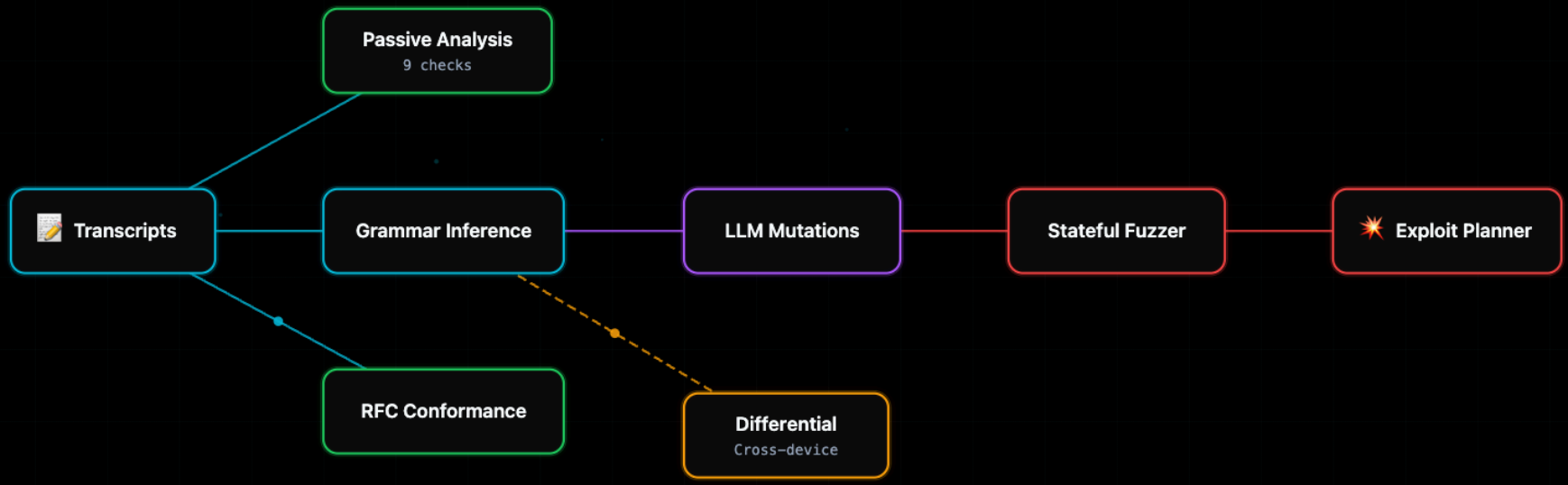
# Analytics Taxonomy

## Five paradigms in protocol security



# Protocol Security Pipeline

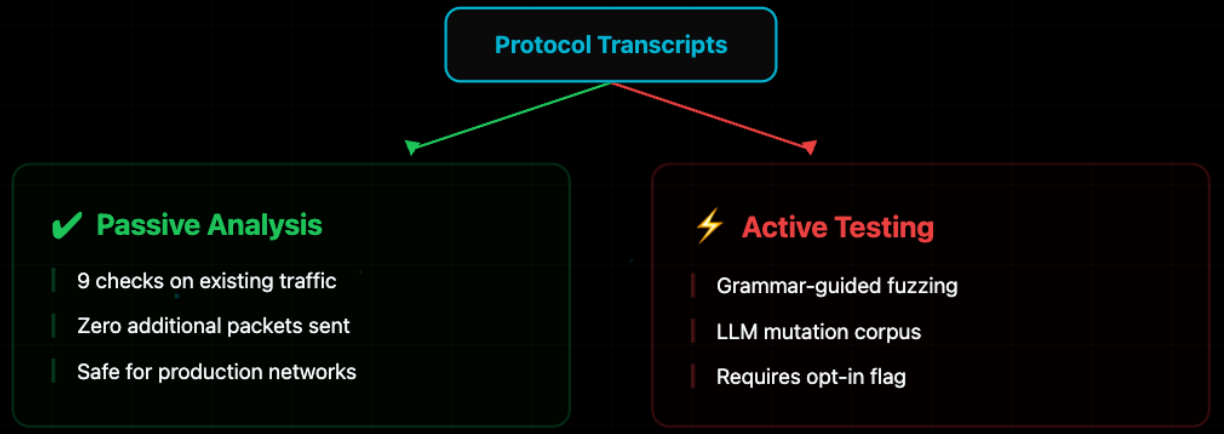
## End-to-end analysis architecture




Transcripts feed three parallel analysis paths

# Passive vs. Active Paths

## Two complementary analysis strategies



# Multiplicative Effect

## Vulnerability propagation across device fleets


**119**
similar devices covered

~~0h~~ → **20m**
assessment time

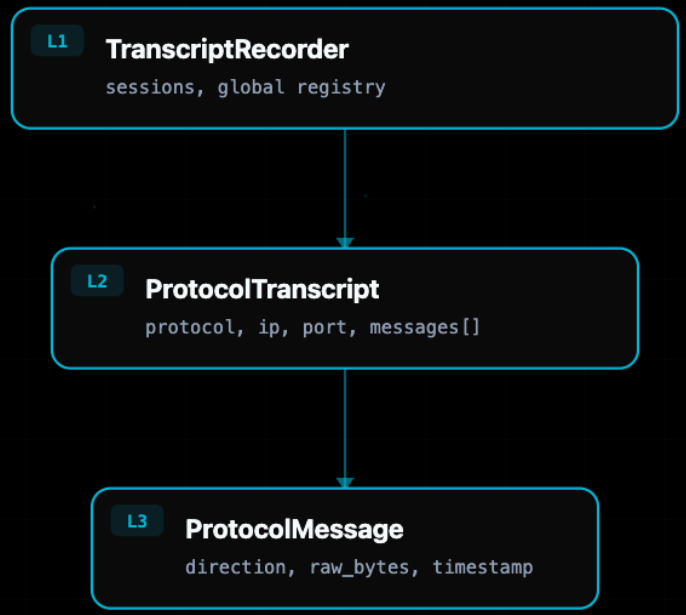
SECTION 04

# Protocol Transcript Recording

Application-layer capture without pcap or NET_RAW

# Transcript Data Model

## Three-layer recording architecture



# ProtocolMessage Dataclass

## The atomic unit of protocol recording

protocol_schemas.py — PYTHON

```python
class Direction(str, Enum):  # ← Direction enum
    SENT = "sent"
    RECEIVED = "received"

@dataclass
class ProtocolMessage:
    direction: Direction  # ← SENT or RECEIVED
    raw_bytes: bytes
    timestamp: float
    metadata: dict[str, Any] = field(
        default_factory=dict
    )
```

Each message captures direction, raw bytes, and timing for full replay capability

# Dual Representation

## Both hex and text views stored per message

**HEX VIEW**

```
48 54 54 50 2f 31 2e 31
20 32 30 30 20 4f 4b 0d
0a 43 6f 6e 74 65 6e 74
2d 54 79 70 65 3a 20 74
65 78 74 2f 68 74 6d 6c
```

**TEXT VIEW**

```
HTTP/1.1 200 OK
Content-Type: text/html
Server: lighttpd/1.4.35
Content-Length: 2048
Connection: keep-alive
```

Both stored for dashboard display and programmatic analysis

# Response Fingerprint

## SHA-256 hash as device behavior identity

protocol_transcript.py — PYTHON

```python
def response_fingerprint(self) -> str:
    """SHA-256 of all received bytes."""
    h = hashlib.sha256()  ← SHA-256 accumulator
    for m in self.messages:
        if m.direction == Direction.RECEIVED:  ← Only RECEIVED bytes
            h.update(m.raw_bytes)
    return h.hexdigest()  ← 64-char hex output
```

**RESULT**

`a3f2e8b1c4d596... (64-char hex)`

= device behavior identity

# Transcript Recorder API

## Context manager captures protocol exchanges

protocol_transcript.py · 287 lines

protocol_transcript.py PYTHON

```python
recorder = TranscriptRecorder()

with recorder.session("http", ip, 80) as tx:  ← Opens protocol session context
    tx.record_sent(request_bytes)  ← Timestamps + sizes recorded
    tx.record_received(response_bytes)  ← Latency computed automatically

# Empty sessions silently discarded  ← Zero-length sessions are dropped
```

No pcap needed | Works in userspace | Thread-safe via GIL

# Design Decision: No pcap

## Application-layer recording over raw packet capture

### ✘ pcap / libpcap

- **Requires root / NET_RAW**
  Privilege escalation barrier
- **Captures everything (noisy)**
  Background traffic floods data
- **Below TLS (ciphertext only)**
  Cannot see decrypted payload

VS

### ✔ App-Layer Recording

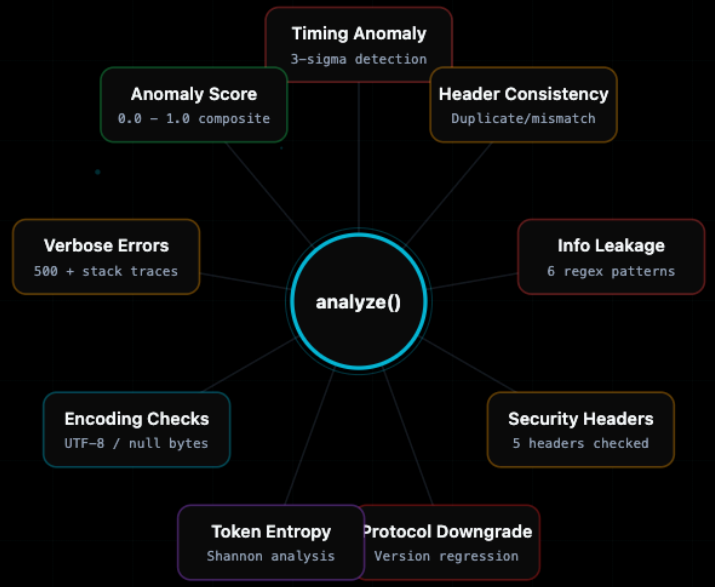
- **No privileges needed**
  Runs as normal user process
- **Only our traffic**
  Scoped to scan-initiated flows
- **Decrypted payload**
  Records after TLS termination

**App-layer wins for security tooling**

# Passive Protocol Analyzer

## Nine checks applied to every transcript

passive_analyzer.py · 414 lines



# Timing Anomaly Detection

## 3-sigma statistical analysis of response latencies

### Z-Score Analysis

Step 1: Gather data

1. `Collect response latencies`

Step 2: Compute moments

2. `mean, stddev = statistics()`

Step 3: Normalize

3. `Z = (latency - mean) / stddev`

Step 4: 3-sigma threshold

4. `Flag if Z > 3.0`

**Anomaly Score** `min(z/5, 1.0)`

# Header Consistency Checks

## Detecting duplicate headers and length mismatches

CHECK: DUPLICATE HEADERS

```
HTTP/1.1 200 OK
Content-Type: text/html  DUPLICATE
Content-Type: application/json  DUPLICATE
Server: nginx/1.18.0
```

CHECK: CONTENT-LENGTH MISMATCH

Claimed:  1024 bytes

Actual:  2048 bytes

Severity: HIGH — possible smuggling or buffer overflow

# Information Leakage Detection

## Six regex categories scan every response body

⚠ **Stack Traces** — `Traceback|at \w+\.java:\d+` — HIGH

📂 **File Paths** — `/[\w/]+\.(py|java|php|conf)` — MEDIUM

🌐 **Internal IPs** — `10\.x|172\.(16-31)|192\.168` — HIGH

🔑 **Credentials** — `password|secret|api_key in response` — CRITICAL

🗄 **DB Connections** — `postgres://|mysql://|mongodb://` — CRITICAL

🛠 **Debug Indicators** — `DEBUG=True|X-Debug-Token` — MEDIUM

Each match contributes 0.7 to anomaly score

# Protocol Downgrade Detection

## Version regression across protocol versions


cf. POODLE attack (CVE-2014-3566) — forced TLS 1.0 downgrade

# Session Token Entropy

## Shannon entropy measures token randomness quality

### Shannon Entropy

1. Entropy formula

```
H(X) = -SUM p(x) log2(p(x))
```

2. Minimum acceptable

```
Threshold: 3.5 bits/char
```

3. Good randomness

```
Hex random: ~4.0 bits/char
```

4. Predictable (counter)

```
Sequential: ~0 bits/char
```

**Low entropy** — **Severity 0.7**

EXAMPLE TOKENS

| | | |
|---|---|---|
| a3f7b2e91c0d8456 | 4 bits | GOOD |
| 0001000200030004 | 1.2 bits | WEAK |
| admin | 2.3 bits | BAD |

Tokens < 8 chars = immediate HIGH severity

# Passive Analysis Summary

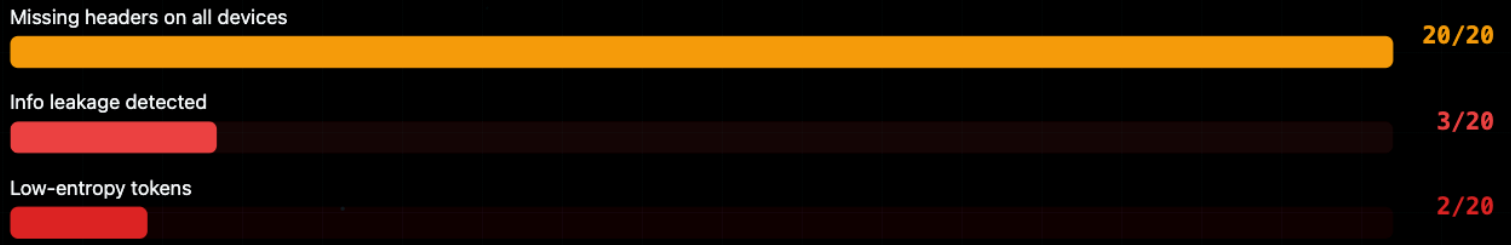
Nine checks, zero additional network traffic


Passive Checks

**0ms overhead**
Constant Time

**0 bytes**
Extra Traffic

IOT SIMULATION RESULTS



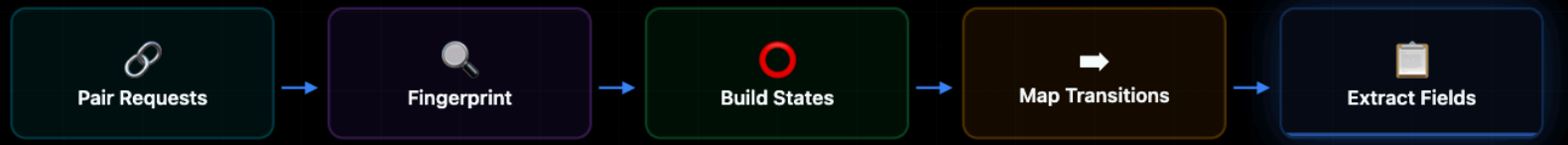
SECTION 05

# Grammar Inference: DynPRE

Automatically extracting protocol state machines from network traffic

# Grammar Extraction Pipeline

## Five-stage process from raw transcripts to protocol grammar




```
Entire pipeline runs in <100ms per protocol transcript
```

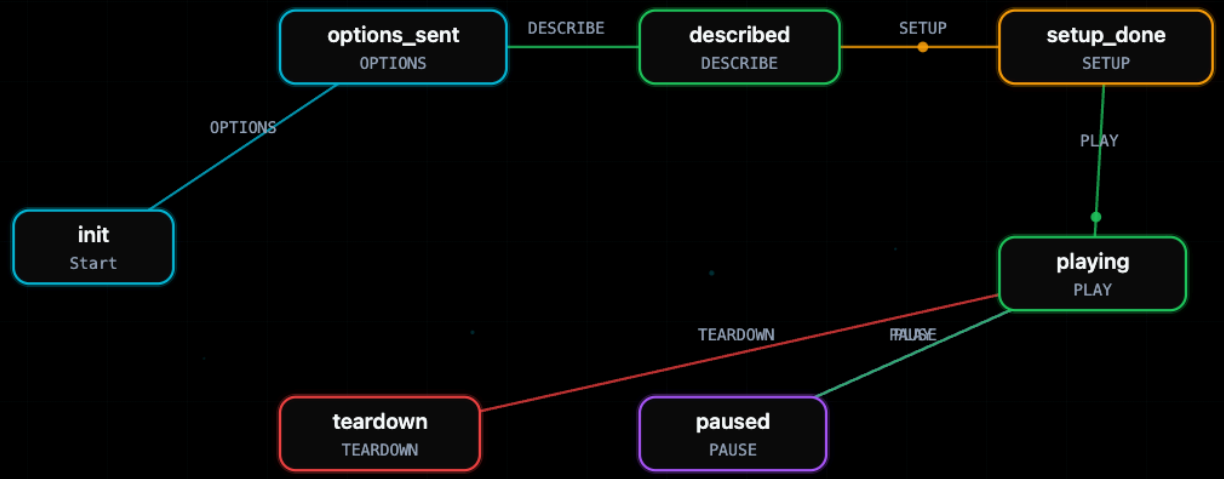
# State Machine Construction

## Building minimal FSM from fingerprinted responses


Duplicate transition paths merged → minimal FSM

# RTSP Seed State Diagram

## Seven states covering the full streaming lifecycle




RFC 7826 compliant — PLAY/PAUSE bidirectional transitions

SECTION 06

# RFC Conformance Scoring

Measuring implementation quality against protocol specifications

# Conformance Scoring

Simple ratio of rules passed to rules checked

## Score Derivation

Step 1: Count total rules

1. `total = count(rules_checked)`

Step 2: Count passing rules

2. `passed = count(rules_passed)`

Step 3: Compute percentage

3. `Score = (passed / total) x 100`

| Score Range | 0 – 100% |
|---|---|

WORKED EXAMPLE

| | |
|---|---|
| Rules checked: | 18 |
| Rules passed: | 15 |
| **Score:** | **83%** |

More protocols = more rules = each deviation costs less

# HTTP Conformance Rules

## Five checks against RFC 9110 (HTTP Semantics)

| | | | |
|---|---|---|---|
| 1 | **Status Line Format**<br>HTTP/x.y NNN Reason-Phrase | RFC 9110 sec 15.1 | HIGH |
| 2 | **Date Header**<br>Origin servers SHOULD send Date | RFC 9110 sec 6.6.1 | INFO |
| 3 | **Content-Length Consistency**<br>Must match actual body size | RFC 9110 sec 8.6 | MEDIUM |
| 4 | **Server Version Disclosure**<br>Revealing version aids attackers | RFC 9110 sec 10.2.4 | LOW |
| 5 | **Allow Header Methods**<br>Listed methods must be valid | RFC 9110 sec 10.2.1 | LOW |

# RTSP Conformance Rules

## Three checks against RFC 7826 (RTSP 2.0)

1 **Status Line Format** HIGH sec 7.1
RTSP/x.y NNN format required
Must start with RTSP version identifier

2 **CSeq Header Presence** MEDIUM sec 18.20
Every request/response must carry CSeq
Matches responses to requests in pipeline

3 **Public Header in OPTIONS** LOW sec 13.1
OPTIONS response lists supported methods
Informs client of server capabilities

```
CSeq: 3                                  CSeq: 3
Request: DESCRIBE rtsp://cam/stream  ──►  Response: RTSP/1.0 200 OK   Matched by sequence number
```

# Non-Conformance Predicts Vulnerabilities

## Low RFC score correlates with more fuzzer findings


**3x**
more vulnerabilities below score 70

**21**
rules across 6 protocols

SECTION 07

# Differential Analysis

Fleet-wide anomaly detection across similar devices

# Differential Analysis Scenario

## Same vendor, same model, different behavior

Vendor: Hikvision DS-2CD2132

**Cam A**

GET /onvif/device

✓ identical

**Cam B**

GET /onvif/device

✓ identical

**Cam C**

GET /onvif/device

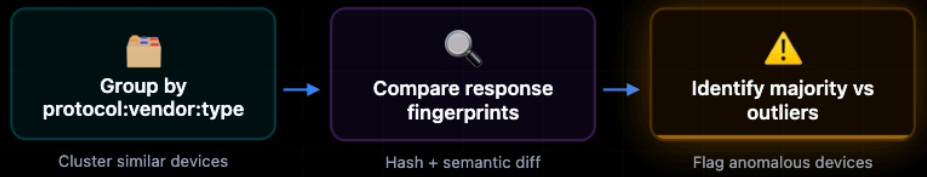
✗ different

**Why is Cam C different?**

# Differential Analysis Algorithm

## Three-stage comparison pipeline

ANALYSIS PIPELINE




POSSIBLE OUTLIER CAUSES

- Different firmware
- Misconfiguration
- Compromise
- Counterfeit device

# Anomaly Scoring

## Weighted deviation with minority factor

1. Severity weights for each diff level

```
weights: high=0.4, medium=0.2, low=0.1
```

2. Accumulate weighted deviations

```
deviation_score = sum(weights), capped at 1.0
```

3. Penalize large anomaly groups less

```
minority_factor = 1 - (anomaly_size / total)
```

4. Combined anomaly score

```
final = deviation_score x minority_factor
```

where example = 0.4 x 0.91 = 0.364

| 1 of 10 outlier with Server header diff | 0.364 |
|---|---|

`deviation=0.4 (Server header) | minority=1 - 1/10 = 0.91 | final = 0.364`

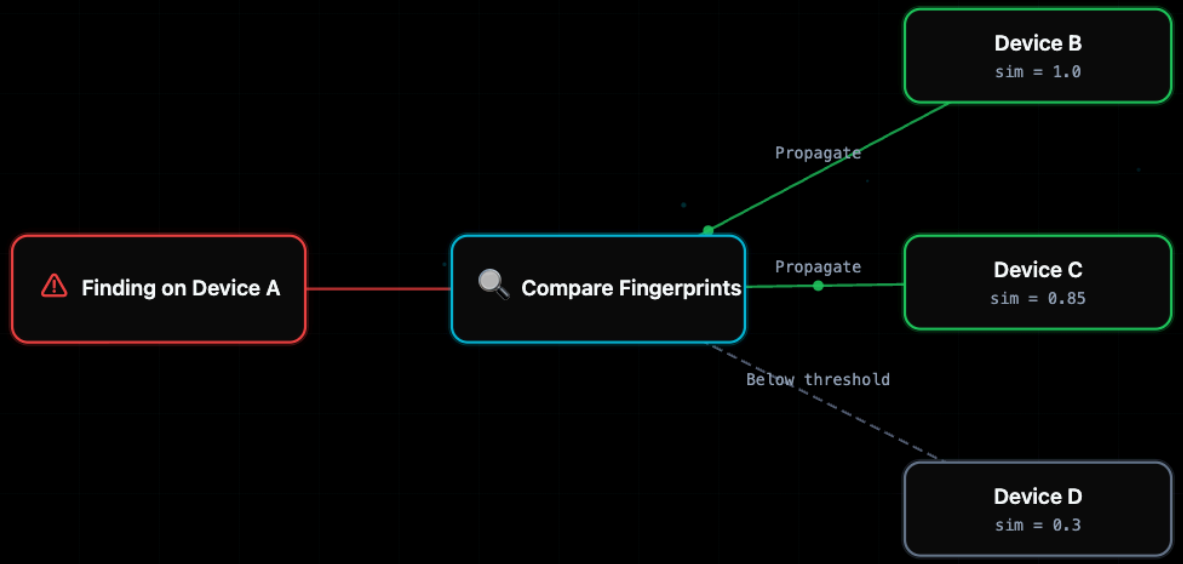
SECTION 07

# Vulnerability Propagation

Fleet-wide intelligence from individual findings

# Propagation Algorithm

## Finding fan-out based on fingerprint similarity




Similarity threshold: 0.50 (configurable)

# Real-World: Hospital Camera Fleet

## Propagation eliminates redundant fuzzing

**200** IP cameras

120 Hikvision + 80 Dahua

Without Propagation — Fuzz all 200 devices

**33 hours**

With Propagation — Fuzz 2 representatives

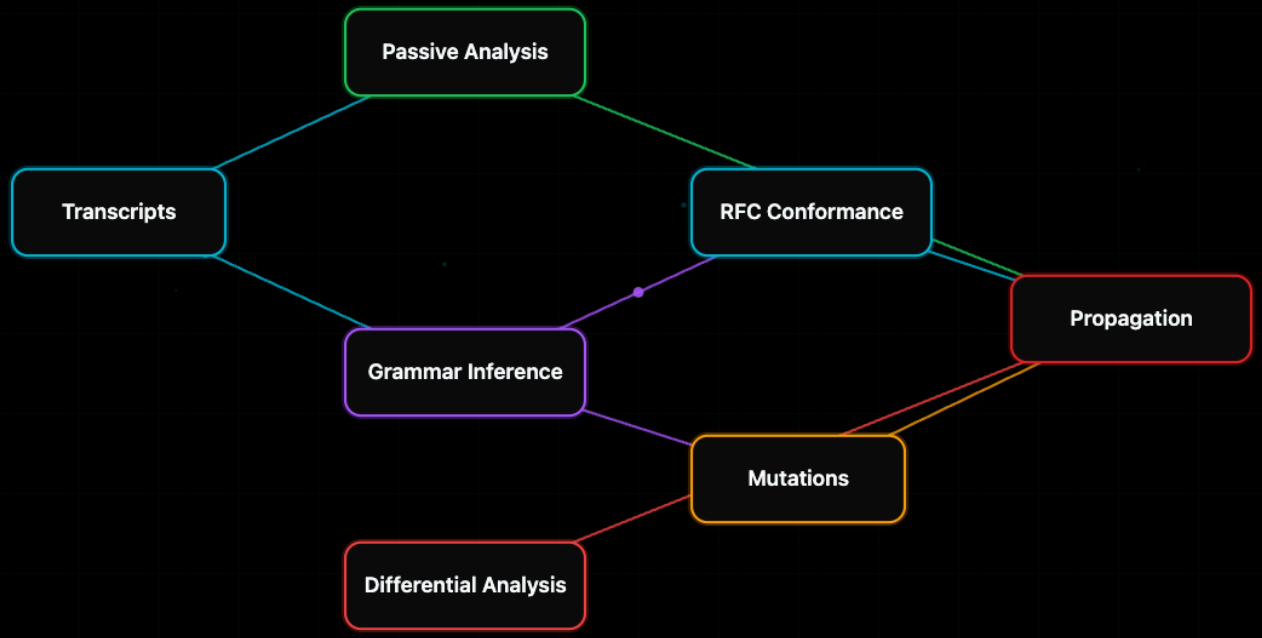
**20 min**

**99x** faster

Linear \u2192 near-constant time

# Phase 3 Analytical DAG

## Directed acyclic graph of analysis stages




Multiplicative finding growth: `1 transcript` `\u2192` **`47 findings`**

SECTION 09

# Attack Chain Planning

Multi-protocol exploit path construction and scoring

# Real-World: Mirai Botnet Chain

## Two-step infection using default credentials

🤖 **MIRAI BOTNET** 2016 — 600k devices


Sophisticated IoT attacks use 3-4 steps

Our engine models multi-step chains

# Feasibility Scoring

## Quantifying attack chain likelihood

### Chain Feasibility Formula

Start with maximum feasibility

1. `Base = 1.0`

High confidence — observed in traffic

2. `Evidence-backed step: × 0.9`

Low confidence — inferred from model

3. `Speculative step: × 0.5`

Boost if credentials confirmed

4. `Default creds present: × 1.5 (cap 1.0)`

**WORKED EXAMPLE: SSDP \u2192 HTTP \u2192 RCE on Hikvision**

`0.9 × 0.5 × 0.5 = 0.225` | Severity: 10.0 (RCE)

**0.225 × 10.0 → Warrants investigation**

PHASE 3 COMPLETE

# Protocol Security Engine & Zero-Day Discovery

NEXT

## Phase 4: Predictive Attack Path Analytics

Dr. Mallarapu · SEAS-8414 · Breakwater Security Platform