Independent Submission
Request for Comments: XXXX
Category: Standards Track
ISSN: XXXX-XXXX

B. Kreinz
TLCTC Framework
December 2025

# Top Level Cyber Threat Clusters (TLCTC)
## Version 2.0 Specification

**Abstract**

This document specifies Top Level Cyber Threat Clusters (TLCTC), a cause-oriented, actor-agnostic framework that establishes a consistent language for describing cyber risk through ten non-overlapping threat clusters spanning human, physical, and digital domains. Each cluster classifies a distinct attack vector according to the generic vulnerability it initially targets, enabling sequential expression of real attack paths from initial compromise through follow-on steps without conflating threats with outcomes such as data loss, fraud, or service disruption.

TLCTC separates a strategic management view (cluster-level risk and generic vulnerabilities) from an operational security view (concrete vulnerabilities, techniques, and procedures), creating a stable backbone for governance, control design, threat intelligence exchange, and incident learning. This specification defines the canonical cluster definitions, classification grammar, attack path notation, velocity annotations, domain boundary operators, and JSON interchange format required for conformant implementations.

## Status of This Memo

This document specifies a Standards Track specification for the cybersecurity community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

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

## Copyright Notice

**Table of Contents**

# 1. Introduction

## 1.1. Problem Statement

Cybersecurity suffers from a persistent language problem: practitioners describe fundamentally different things using the same words, and use different words for the same thing. The term "cyber threat" is routinely conflated with threat actors, vulnerabilities, control failures, incidents, and outcomes (e.g., data breach, denial of service, ransomware). This semantic blur makes it difficult to compare cyber incidents, aggregate threat intelligence, design targeted controls, or communicate risk consistently between leadership, risk functions, and technical teams.

A comprehensive analysis of existing cybersecurity standards reveals a critical gap: the absence of a unified, cause-oriented threat taxonomy that enables consistent risk assessment across organizations and sectors. Standards bearing "Cyber" in their names—from ISO/IEC 27001 to NIST CSF 2.0, CMMC 2.0, and EU regulations—provide control frameworks and risk management processes but do not define what threats are or how to classify them consistently.

## 1.2. Solution Overview

Top Level Cyber Threat Clusters (TLCTC) addresses this foundational deficit by anchoring cyber analysis in causality: a cyber threat is defined by the generic vulnerability (root weakness) it exploits, not by who performs it and not by what consequence follows.

TLCTC is built on a strict classification rule: every attack step exploits exactly one generic vulnerability, and each generic vulnerability belongs to exactly one of ten non-overlapping clusters. Each attack vector is therefore defined by the generic vulnerability it initially targets, and complete real-world intrusions can be represented as attack paths—ordered sequences of cluster steps—without changing the meaning of the individual steps.

## 1.3. Document Structure

This specification is organized as follows: Section 2 defines terminology and normative keywords. Section 3 establishes the foundational axioms. Section 4 provides canonical cluster definitions. Section 5 defines the classification grammar. Section 6 specifies attack path notation. Section 7 defines attack velocity measurements. Section 8 specifies domain boundary operators. Section 9 defines the JSON interchange architecture. Section 10 establishes conformance requirements.

# 2. Terminology

## 2.1. Normative Keywords

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

When these keywords appear in lowercase, they carry their ordinary English meaning.

## 2.2. Definitions

### 2.2.1. Core Terms

**Attack Step:** A single attacker action or event that exploits exactly one generic vulnerability in a specific context. Each Attack Step MUST map to exactly one TLCTC cluster.

**Attack Path:** An ordered sequence of Attack Steps representing a complete attack scenario. Basic notation: #X → #Y → #Z.

**Attack Vector:** A distinct initiating method defined by the initial generic vulnerability targeted.

**Attack Velocity (Δt):** The time interval between two adjacent Attack Steps in an attack path.

**Generic Vulnerability:** A root weakness category that defines a TLCTC cluster. Each of the 10 clusters is defined by exactly one generic vulnerability.

**Threat Cluster:** One of the 10 TLCTC categories, each defined by the generic vulnerability it addresses.

### 2.2.2. Outcome Terms

**Data Risk Event (DRE):** An outcome event describing Loss of Confidentiality (C), Loss of Integrity (I), or Loss of Availability (A) for data. DREs are recorded separately from cluster steps.

**Loss of Control (LoC):** The central event in the Bow-Tie model representing the point at which the attacker achieves unauthorized control over the system.

### 2.2.3. Execution Terms

**Exploit Code:** Foreign code/payload crafted to trigger implementation flaws in software (#2 or #3), forcing unintended data→code transitions.

**Malware Code:** Foreign executable content that executes via the environment's designed execution capabilities (#7). Uses intended execution paths.

**Foreign Executable Content (FEC):** In TLCTC v2.0, specifically refers to Malware Code that executes in #7.

### 2.2.4. Topology Terms

**Domain:** A set of assets governed by a coherent control regime.

**Responsibility Sphere:** The organizational owner of a domain, denoted as @Entity.

**Domain Boundary:** A point where responsibility spheres or control regimes change.

**Bridge Cluster:** A cluster whose generic vulnerability inherently enables attacks to cross domain boundaries.

**Internal Cluster:** A cluster that operates within a single domain's control regime.

# 3. Foundational Axioms

TLCTC is a precise, cause-oriented language for describing cyber threats. The following axioms define foundational premises that constrain interpretation and ensure consistent reasoning.

## 3.1. Scope Axioms

### 3.1.1. Axiom I — No System-Type Differentiation

TLCTC applies to generic IT assets and their context and therefore does not differentiate by IT system type. Sector labels (e.g., SCADA, IoT, cloud, medical devices, network appliances) do not create new threat classes; they only change the specific vulnerabilities and controls at the operational level.

### 3.1.2. Axiom II — Client-Server Universal Model

Any networked system interaction can be modeled as client-server (caller-called) interaction at one or more layers. The TLCTC clusters address the generic vulnerabilities that arise from these interactions, independent of protocol or architecture depth.

## 3.2. Separation Axioms

### 3.2.1. Axiom III — Threats Are Causes, Not Outcomes

Threat clusters are on the cause side of the Bow-Tie model. They MUST NOT be conflated with outcomes (data risk events) such as Loss of Confidentiality, Loss of Integrity, or Loss of Availability.

### 3.2.2. Axiom IV — Threats Are Not Threat Actors

Threat clusters are separate from threat actors. Actor identity (attribution, motivation, capability) is not a structuring element for threat categorization; TLCTC classifies actions and exploited generic vulnerabilities, not "who."

### 3.2.3. Axiom V — Control Failure Is Not a Threat

Control failure is control-risk (deviation from a control objective) and MUST NOT be treated as a threat category. Risk remains structured as Threat → Event/Incident → Consequences.

## 3.3. Classification Axioms

### 3.3.1. Axiom VI — One Step, One Generic Vulnerability, One Cluster

Every distinct attack step exploits exactly one generic vulnerability (root weakness) in the attack surface of the target asset. Each generic vulnerability maps to exactly one TLCTC threat cluster.

### 3.3.2. Axiom VII — Attack Vectors Defined by Initial Generic Vulnerability

Each distinct attack vector is defined by the generic vulnerability it initially targets. Classification is anchored in this initial cause, not in technique labels or downstream effects.

### 3.3.3. Axiom VIII — Strategic vs Operational Layering

Each Top-Level Threat Cluster encompasses operational sub-threats, separating a stable Strategic Management Layer (clusters / generic vulnerabilities) from an Operational Security Layer (specific vulnerabilities, techniques, and procedures).

## 3.4. Sequence Axioms

### 3.4.1. Axiom IX — Clusters Chain into Attack Paths

Top-Level Threat Clusters are chained into attack paths to represent complete scenarios (including lateral movement and parallel steps). The time between successive cluster steps is a scenario attribute ($\Delta t$); the set of $\Delta t$ values expresses the attack velocity of the path.

### 3.4.2. Axiom X — Credentials Have Dual Operational Nature

Credential acquisition maps to an enabling cluster (determined by the generic vulnerability used to obtain the credential); credential application (use) always maps to #4 Identity Theft.

# 4. Canonical Cluster Definitions

This section provides the canonical definitions for all ten TLCTC clusters. Each definition includes: Definition, Generic Vulnerability, Attacker's View, Developer's View, Boundary Tests, and Topology classification.

## 4.1. #1 Abuse of Functions

**Definition:** Manipulation of legitimate software capabilities—features, APIs, configurations, administrative settings, workflows—through standard interfaces using built-in input types and valid sequences of actions. The step achieves an attacker advantage without requiring an implementation flaw.

**Generic Vulnerability:** The inherent trust, scope, and complexity designed into software functionality and configuration.

**Attacker's View:** "I abuse a functionality, not a coding issue."

**Developer's View:** "I must understand and constrain the functional domain of my code."

**Boundary Tests (Normative):** If an implementation flaw is required → #2 or #3. If this step enables execution of FEC → record #1 for enablement and → #7 for execution.

**Topology:** Internal.

## 4.2. #2 Exploiting Server

**Definition:** Triggering an implementation flaw in server-role software using Exploit Code, exploiting coding mistakes in how the server processes requests, handles data, enforces logic, or manages resources. This forces an UNINTENDED data→code transition.

**Generic Vulnerability:** Exploitable flaws within server-side source code implementation and its resulting logic.

**Attacker's View:** "I abuse a flaw in the application's source code on the server side."

**Developer's View:** "I must apply secure coding principles for all server-side code."

**Boundary Tests (Normative):** If no implementation flaw → #1. If vulnerable component is client-role → #3. If exploitation results in FEC execution → append #7.

**Topology:** Internal.

## 4.3. #3 Exploiting Client

**Definition:** Triggering an implementation flaw in client-role software through crafted content/responses/state, exploiting coding mistakes in parsing, rendering, state management, or response handling.

**Generic Vulnerability:** Exploitable flaws within client-role source code implementation.

**Attacker's View:** "I abuse a flaw in the source code of software acting as a client."

**Developer's View:** "I must apply secure coding principles for client-role code and never trust incoming data."

**Boundary Tests (Normative):** If no implementation flaw → #1. If vulnerable component is server-role → #2. If exploitation results in FEC execution → append #7.

**Topology:** Internal.

## 4.4. #4 Identity Theft

**Definition:** Presentation/use of credentials, tokens, keys, session artifacts, or other identity representations to authenticate and act as an identity different from the presenter's own.

**Generic Vulnerability:** Weak binding between identity and authentication artifacts, combined with insufficient credential and session lifecycle controls.

**Attacker's View:** "I abuse credentials to operate as a legitimate identity."

**Developer's View:** "I must implement secure credential lifecycle management."

**Boundary Tests (Normative):** Credential acquisition maps to the enabling cluster; credential use always maps to #4 (R-CRED).

**Topology:** Internal.

## 4.5. #5 Man in the Middle

**Definition:** Exploitation of a controlled position on a communication path through interception, observation, modification, injection, replay, or protocol downgrade/stripping.

**Generic Vulnerability:** Insufficient end-to-end confidentiality/integrity protection and implicit trust in local networks and intermediate path infrastructure.

**Attacker's View:** "I abuse my position between communicating parties."

**Developer's View:** "I must ensure confidentiality and integrity of data in transit."

**Boundary Tests (Normative):** Gaining the position maps to another cluster; #5 begins once the position is controlled (R-MITM).

**Topology:** Internal.

## 4.6. #6 Flooding Attack

**Definition:** Exhaustion of finite system resources (bandwidth, CPU, memory, storage, quotas, pools) through volume or intensity that exceeds capacity limits.

**Generic Vulnerability:** Finite capacity limitations inherent in any system component.

**Attacker's View:** "I abuse the circumstance of always limited capacity in systems."

**Developer's View:** "I must implement efficient resource management."

**Boundary Tests (Normative):** If availability loss is primarily caused by an implementation defect → #2/#3. If primarily capacity exhaustion → #6 (R-FLOOD).

**Topology:** Internal.

## 4.7. #7 Malware

**Definition:** Execution of Foreign Executable Content (FEC) through the environment's designed execution capabilities (binaries, scripts, macros, modules, or attacker-controlled commands fed into interpreters).

**Generic Vulnerability:** The environment's intended capability to execute potentially untrusted executable content.

**Attacker's View:** "I abuse the environment's designed capability to execute malware code."

**Developer's View:** "I must control execution paths: allow-listing, code signing, sandboxing."

**Boundary Tests (Normative):** If FEC executes → #7 (per R-EXEC), even if execution is in-memory. If legitimate function misuse enables FEC → #1 → #7.

**Topology:** Internal.

## 4.8. #8 Physical Attack

**Definition:** Unauthorized physical interaction with or interference to hardware, facilities, media, interfaces, or signals—via direct contact or exploitation of physical phenomena/emanations.

**Generic Vulnerability:** Physical accessibility of infrastructure and the exploitability of physical-layer properties.

**Attacker's View:** "I abuse the physical accessibility or properties of hardware and devices."

**Developer's View:** "I must assume physical access can mean compromise."

**Boundary Tests (Normative):** If the physical step leads to FEC execution → #8 → #7.

**Topology:** Bridge (Physical → Cyber).

## 4.9. #9 Social Engineering

**Definition:** Psychological manipulation that causes a human to perform an action counter to security interests—disclosing information, granting access, executing content, modifying configuration, or bypassing procedures.

**Generic Vulnerability:** Human psychological factors (trust, fear, urgency, authority bias, curiosity, ignorance, fatigue).

**Attacker's View:** "I abuse human trust and psychology to deceive individuals."

**Developer's View:** "I must design interfaces and processes that promote secure behavior."

**Boundary Tests (Normative):** Technical vulnerabilities (CVEs) are never #9. #9 is only the human manipulation step.

**Topology:** Bridge (Human → Cyber).

## 4.10. #10 Supply Chain Attack

**Definition:** Exploitation of an organization's third-party trust link such that the organization accepts third-party-originating artifacts or decisions as authoritative within the organization's domain.

**Generic Vulnerability:** Necessary reliance on, and implicit trust placed in, external suppliers/services whose security posture is outside direct organizational control.

**Attacker's View:** "I abuse the target's trust in third parties they rely on."

**Developer's View:** "I must minimize third-party trust, harden trust-acceptance points, verify provenance."

**Boundary Tests (Normative):** Place #10 at the Trust Acceptance Event (TAE) where the third-party trust link is honored.

**Topology:** Bridge (Third-party → Organization).

# 5. Classification Grammar

## 5.1. Global Mapping Rules (R-* Rules)

These rules are global and normative. When classifying any Attack Step, these rules MUST be consulted and applied where relevant.

### 5.1.1. R-ROLE — Server vs Client Determination

If the vulnerable component accepts and handles inbound requests relative to the attacker, the step MUST be classified as #2. If the vulnerable component consumes external responses or content relative to the attacker, the step MUST be classified as #3.

### 5.1.2. R-CRED — Credential Lifecycle Non-Overlap

Credential acquisition maps to the enabling cluster. Credential application MUST always map to #4 Identity Theft. If both occur in the same scenario, they MUST be represented as at least two steps.

### 5.1.3. R-MITM — Position vs Action

The method of gaining a privileged communication-path position maps to another cluster. #5 begins only once the attacker controls a point on the communication path and performs MitM actions.

### 5.1.4. R-FLOOD — Capacity vs Defect

If the primary mechanism is capacity exhaustion by volume/intensity → #6. If the primary mechanism is a defect-triggered crash/degradation (including algorithmic complexity) → #2 or #3.

### 5.1.5. R-EXEC — FEC Execution Recording

If Foreign Executable Content executes, #7 MUST be recorded as its own step at the moment of execution, in addition to the enabling step.

### 5.1.6. R-SUPPLY — Trust Acceptance Event Placement

#10 MUST be placed at the Trust Acceptance Event (TAE) where the third-party trust link is honored and becomes authoritative inside the organization.

### 5.1.7. R-HUMAN — Human Manipulation

If the attacker's advantage comes from human psychological manipulation, #9 MUST be recorded, and subsequent technical steps MUST be separate.

### 5.1.8. R-PHYSICAL — Physical Access

If the attacker's advantage comes from physical access/interference, #8 MUST be recorded, and subsequent technical steps MUST be separate.

### 5.1.9. R-ABUSE — Function Misuse

If no implementation flaw is required and the attacker abuses intended functionality via standard interfaces, the step MUST be #1.

## 5.2. Tie-Breaker Precedence Rules

When multiple clusters seem applicable, apply these precedence rules in order:

| Precedence | Rule |
|---|---|
| 1 | Classify by Initial Generic Vulnerability (not outcomes, actors, tools) |
| 2 | Implementation Flaw (#2/#3) vs Legitimate Function Misuse (#1) |
| 3 | Credential Use Always Wins for the Use Step (#4) |
| 4 | MitM Starts at Controlled Position (#5) |
| 5 | Flooding Is About Capacity; Defects Are #2/#3 |
| 6 | FEC Execution Must Be Explicit (#7) |
| 7 | Human/Physical/Third-Party Are Not Shortcuts |
| 8 | Document Non-Obvious Decisions |

# 6. Attack Path Notation

This section defines the syntax and semantics for expressing TLCTC attack sequences as compact, shareable strings.

## 6.1. Basic Notation Elements

### 6.1.1. Cluster References

**Strategic Layer:** #X where X ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

**Operational Layer:** TLCTC-XX.YY where XX is zero-padded cluster (01-10) and YY is sub-cluster (00-99)

**Equivalence:** #X and TLCTC-0X.00 are semantically equivalent.

### 6.1.2. Sequence Operator

Notation: → (or -> for ASCII compatibility)

Represents temporal progression: #X → #Y means step #X occurs, then step #Y occurs.

### 6.1.3. Parallel Operator

Notation: (#X + #Y)

Indicates two or more clusters occurring simultaneously or in tight coordination within the same attack phase.

### 6.1.4. Domain Boundary Operator

Notation: |||[context][@Source→@Target]|||

Used to explicitly mark where an attack path crosses responsibility spheres.

Components:

[context] — the channel or mechanism (e.g., dev, update, auth, physical, human)

[@Source→@Target] — the responsibility spheres (e.g., @Vendor→@Org)

### 6.1.5. Data Risk Event Tags

Notation: + [DRE: X] where X ∈ {C, I, A} or combinations

Records outcomes separately from cluster steps. DREs MUST NOT be represented as standalone nodes.

## 6.2. Formal Grammar (ABNF)

```
PATH        = PHASE *(SP* EDGE SP* PHASE)
EDGE        = (ARROW / ARROW_ASCII) [SP* DT_ANN]
ARROW       = "→"
ARROW_ASCII = "->"
PHASE       = STEP / PAR_GROUP
PAR_GROUP   = "(" SP* STEP *(SP* "+" SP* STEP) SP* ")"
STEP        = CLUSTER_REF [SP* BOUNDARY] *(SP* STEP_ANN) [SP* DRE_TAG]
CLUSTER_REF = STRATEGIC / OPERATIONAL
STRATEGIC   = "#" ( "10" / "1" / "2" / ... / "9" )
```

```
OPERATIONAL = "TLCTC-" 2DIGIT "." 2DIGIT
```

## 6.3. Notation Examples

**Phishing to credential use:** #9 → #4

**Server exploit to execution:** #2 → #7

**Supply chain with boundary:** #10 ||[dev][@Vendor→@Org]|| → #7

**Parallel execution:** #4 → (#1 + #7)

**With velocity:** #9 →[Δt=2h] #4 →[Δt=5m] #1

**With DRE:** #2 + [DRE: C]

# 7. Attack Velocity (Δt)

## 7.1. Definition

Attack Velocity (Δt) is the time interval between two adjacent Attack Steps in an attack path. For an edge #X → #Y, the value Δt(X→Y) represents the elapsed time between step #X and step #Y.

Δt is an edge property attached to the sequence operator, not to steps themselves.

## 7.2. Duration Notation

### 7.2.1. Canonical Duration Format

Format: <number><unit> where unit is one of: ms, s, m, h, d, w, mo, y

Examples: Δt=0s, Δt=12m, Δt=24h, Δt=7d

### 7.2.2. Modifiers

**Approximate:** Δt~15m

**Upper bound:** Δt<15m

**Lower bound:** Δt>15m

**Range:** Δt=10m..20m

**Unknown:** Δt=?

**Instant:** Δt=instant (corresponds to Δt≤1s)

## 7.3. Velocity Classes (Normative)

Δt values are grouped into four operational velocity classes describing defender response feasibility:

| Class | Typical Δt | Threat Dynamics | Primary Defense Mode |
|---|---|---|---|
| VC-1: Strategic | Days → Months | Slow transitions, long dwell | Log retention, threat hunting |
| VC-2: Tactical | Hours | Human-operated | SIEM alerting, analyst triage |
| VC-3: Operational | Minutes | Automatable | SOAR/EDR automation |
| VC-4: Real-Time | Seconds → ms | Machine-speed | Architecture, circuit breakers |

## 7.4. Interpretation Rules

If a critical transition is VC-3 or faster, purely human response is structurally insufficient. Controls MUST be automated or architectural.

Velocity classes are operational lenses, not threat categories. They MUST NOT be used to replace clusters.

# 8. Domain Boundaries and Topology

## 8.1. Topology Classification

TLCTC clusters are classified into two topology types based on whether they cross domain boundaries.

### 8.1.1. Bridge Clusters

Clusters whose generic vulnerability resides outside the software domain and commonly serve as responsibility-sphere transition pivots:

#8 Physical Attack (Physical → Software)

#9 Social Engineering (Human → IT)

#10 Supply Chain Attack (Third-party → Organization)

### 8.1.2. Internal Clusters

Clusters that operate primarily within the software domain's attack surfaces:

#1 Abuse of Functions

#2 Exploiting Server

#3 Exploiting Client

#4 Identity Theft

#5 Man in the Middle

#6 Flooding Attack

#7 Malware

## 8.2. Domain Boundary Operator

Notation: |||[context][@Source→@Target]||

The operator SHOULD accompany bridge cluster steps and MAY be used with any step that crosses a domain boundary.

### 8.2.1. Context Values

Common context values: dev, update, auth, physical, human, runtime, admin

### 8.2.2. Responsibility Sphere Notation

Format: @Entity or @Entity(Role)

Examples: @Org, @Vendor, @CloudProvider, @Vendor(IdP), @Org(SP)

## 8.3. Examples

**Supply chain:** #10 |||[dev][@Vendor→@Org]|| → #7

**Physical access:** #8 |||[physical][@Facilities→@IT]|| → #7

**Social engineering:** #9 |||[human][@External→@Org]|| → #4

**Federation:** #4 → #10 |||[auth][@Vendor(IdP)→@Org(SP)]|| → #1

# 9. JSON Interchange Architecture

This section specifies the machine-readable interchange format for TLCTC covering static definitions and dynamic instances.

## 9.1. Architecture Overview

The TLCTC JSON architecture is organized into three layers:

### 9.1.1. Layer 1 — Framework Definition (Static)

Defines the immutable dictionary for a TLCTC release: cluster identifiers, definitions, axiom set, and rule identifiers.

**Schema:** tlctc-framework.schema.json

**Instance:** tlctc-framework.v2.0.json

### 9.1.2. Layer 2 — Reference Registry (Context)

Defines reusable reference objects: responsibility spheres, allowed boundary operator contexts.

**Schema:** tlctc-reference.schema.json

**Instance:** @Org-registry.vX.Y.Z.json

### 9.1.3. Layer 3 — Attack Path Instances (Dynamic)

Defines specific incidents: sequences, parallel groups, boundary annotations, velocity, outcome tags.

**Schema:** tlctc-attack-path.schema.json

**Instance:** incident-<id>.json

## 9.2. Design Principles (Normative)

### 9.2.1. Separation of Meaning from Measurement

Layer 1 defines meaning (what a cluster is). Layer 3 records measurement (what happened). Incident records MUST NOT re-define clusters.

### 9.2.2. Strict Validation with Explicit Extensions

Schemas use additionalProperties: false. Non-standard fields MUST be placed inside an extensions object.

### 9.2.3. Linear + Parallel Modeling

The incident format models the attack path as an ordered list of sequence_item entries: attack_step for single steps, parallel_group for (#X + #Y) patterns.

## 9.3. Layer 1 Schema Structure

```
{
  "metadata": {
    "tlctc_version": "2.0",
```

```
    "release_date": "2025-12-21",
    "publisher": "TLCTC Framework",
    "license": "CC-BY-4.0"
  },
  "clusters": { "#1": {...}, ... "#10": {...} },
  "axioms": [...],
  "rules": [...]
}
```

## 9.4. Layer 3 Attack Path Instance Structure

```
{
  "metadata": {
    "incident_id": "EXAMPLE-2025-001",
    "tlctc_version": "2.0",
    "framework_ref": "tlctc-framework.v2.0.json"
  },
  "path_sequence": [
    { "step_id": "s1", "cluster": "#9", ... },
    { "step_id": "s2", "cluster": "#7", ... }
  ]
}
```

## 9.5. Conformance Checklist

An incident record is conformant if:

1. It validates against tlctc-attack-path.schema.json
2. metadata.tlctc_version matches the referenced framework version
3. All step_id values are unique within the document
4. If fec_executed=true, the step is #7 or includes fec_recorded_in_step_id pointing to a #7 step
5. All boundary annotations use defined @Sphere identifiers and context labels

# 10. Conformance Requirements

## 10.1. Classification Conformance

A TLCTC classification is conformant if:

1. Every Attack Step maps to exactly one cluster (Axiom VI)
2. Classification is based on initial generic vulnerability (Axiom VII)
3. Outcomes are recorded as DREs, separate from cluster steps (Axiom III)
4. All applicable R-* rules are correctly applied
5. Tie-breaker precedence is followed when multiple clusters seem applicable

## 10.2. Notation Conformance

A TLCTC attack path is conformant if:

1. Contains at least one valid step (cluster reference)
2. Uses only defined operators: → (sequence), + (parallel), ||...|| (boundary)
3. Has balanced parentheses for all parallel groups
4. Does not contain: trailing sequence operators, + outside parentheses, boundary operator as final element

## 10.3. JSON Conformance

A TLCTC JSON document is conformant if it passes validation against the appropriate layer schema and satisfies the Layer 3 conformance checklist (Section 9.5).

# 11. Security Considerations

This specification defines a threat taxonomy and classification system. The following security considerations apply:

## 11.1. Classification Integrity

Consistent classification is essential for threat intelligence sharing. Deviations from the specification reduce comparability and may lead to incorrect control mapping.

## 11.2. Data Sensitivity

Attack path instances (Layer 3) may contain sensitive information about incidents, vulnerabilities, and organizational defenses. Implementers SHOULD apply appropriate access controls and consider TLP markings when sharing.

## 11.3. Validation

Implementers SHOULD validate JSON documents against schemas before processing to prevent injection of malformed data. Layer 3 instances SHOULD be validated for referential integrity (e.g., fec_recorded_in_step_id references).

## 12. IANA Considerations

This document has no IANA actions.

However, implementers establishing registries for TLCTC operational sub-clusters (TLCTC-XX.YY where YY ≠ 00) SHOULD coordinate with the TLCTC community to ensure consistent numbering.

# 13. References

## 13.1. Normative References

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

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

[JSON-SCHEMA] Wright, A., et al., "JSON Schema: A Media Type for Describing JSON Documents", draft-07, 2018.

## 13.2. Informative References

[NIST-CSF] National Institute of Standards and Technology, "Cybersecurity Framework Version 2.0", 2024.

[ISO27001] International Organization for Standardization, "ISO/IEC 27001:2022 Information security management systems", 2022.

[MITRE-ATTACK] MITRE Corporation, "ATT&CK Framework", https://attack.mitre.org/

[STIX] OASIS, "Structured Threat Information Expression (STIX) Version 2.1", 2021.

[FAIR] Factor Analysis of Information Risk, "FAIR Model", https://www.fairinstitute.org/

# Appendix A. Quick Reference Tables

## A.1. Cluster Summary

| # | Name | Generic Vulnerability | Topology |
|---|---|---|---|
| 1 | Abuse of Functions | Functional scope/trust in designed capabilities | Internal |
| 2 | Exploiting Server | Server-side code implementation flaws | Internal |
| 3 | Exploiting Client | Client-side code implementation flaws | Internal |
| 4 | Identity Theft | Identity-artifact binding / credential lifecycle | Internal |
| 5 | Man in the Middle | Lack of end-to-end communication protection | Internal |
| 6 | Flooding Attack | Finite capacity limitations | Internal |
| 7 | Malware | Designed execution capability | Internal |
| 8 | Physical Attack | Physical accessibility/interference | Bridge |
| 9 | Social Engineering | Human psychological factors | Bridge |
| 10 | Supply Chain Attack | Third-party trust dependencies | Bridge |

## A.2. R-* Rules Summary

| Rule | Distinguishes | Key Decision |
|---|---|---|
| R-ROLE | #2 vs #3 | Server-role → #2; Client-role → #3 |
| R-CRED | Acquisition vs Use | Acquisition → enabling cluster; Use → always #4 |
| R-MITM | Gaining vs Exploiting | Gaining position → enabling cluster; Exploiting → #5 |
| R-FLOOD | Capacity vs Defect | Volume exhaustion → #6; Defect → #2/#3 |
| R-EXEC | FEC Execution | If FEC executes → #7 MUST be recorded |
| R-SUPPLY | TAE Placement | #10 at Trust Acceptance Event |
| R-HUMAN | Human Manipulation | Psychological manipulation → #9 |
| R-PHYSICAL | Physical Access | Physical interaction → #8 |
| R-ABUSE | Function Misuse | No flaw, legitimate capability → #1 |

## A.3. Notation Quick Reference

| Element | Notation | Example |
|---|---|---|
| Sequential steps | → | #9 → #4 → #1 |
| Velocity annotation | →[Δt=value] | #9 →[Δt=2h] #4 |
| Parallel steps | (#X + #Y) | (#1 + #7) |
| Domain boundary | \|\|[ctx][@Src→@Tgt]\|\| | #10 \|\|[dev][@Vendor→@Org]\|\| |
| Data Risk Event | + [DRE: X] | #2 + [DRE: C] |
| Strategic cluster | #X | #4 |
| Operational cluster | TLCTC-XX.YY | TLCTC-04.00 |

## Author's Address

Bernhard Kreinz

TLCTC Framework

Website: https://www.tlctc.net

Email: info@tlctc.net