The deliberate architecture behind the TLCTC cluster count — and the refinement space waiting for those with the capacity to fill it.

BK Bernhard Kreinz • February 20, 2026 • 9 min read

When practitioners encounter the Top Level Cyber Threat Clusters framework for the first time, the most common reaction is: *"Why exactly ten?"* Sometimes it's genuine curiosity. Sometimes it's skepticism. Occasionally it's a polite way of saying *"this looks too simple."*

The honest answer has two parts — one analytical, one deliberately provocative.

The 10 clusters are not an arbitrary enumeration. They are the output of a systematic decomposition of the IT landscape into its fundamental aspects, each yielding exactly one class of generic vulnerability. The thought experiment — described in detail in the TLCTC v2.0 whitepaper — imagines the entire IT landscape as a single object and asks: *in how many fundamentally distinct ways can an attacker exploit it?*

The answer falls out of the object's inherent properties: its designed functionality (#1), its server-side code (#2), its client-side code (#3), its identity mechanisms (#4), its communication channels (#5), its finite resource capacity (#6), its code execution capability (#7), its physical form (#8), its human operators (#9), and its dependency on third parties (#10). Each aspect maps to one generic vulnerability. Each generic vulnerability defines one cluster. The decomposition guarantees completeness (no gaps) and mutual exclusivity (no overlaps) — not by convention, but by construction.

So the number 10 is not a design choice. It is a result.

But I would be dishonest if I claimed the number serves only an analytical purpose. It also serves a rhetorical one.

Ten top-level clusters — covering the entire adversarial cyber threat landscape — is compact enough to provoke a reaction. It forces the cybersecurity community to confront a question they have been avoiding for decades: *if ten is wrong, what is the correct number?*

STRIDE offers six categories — but mixes causes with outcomes and cannot represent attack sequences. MITRE ATT&CK provides over 200 techniques — but has no stable strategic layer above them and no consistent causal taxonomy underneath. NIST CSF 2.0 defines six functions for *managing* risk, yet its ID.RA (Risk Assessment) category contains no standardized threat taxonomy to populate it. ENISA publishes annual threat landscapes with categories that shift from year to year.

In other words: the cybersecurity industry has never produced a stable, cause-based answer to the question *"what are the distinct types of cyber threats?"* — a question that logically precedes every control selection, every risk assessment, and every regulatory mandate.

The number 10 is an invitation to do that work. Or to adopt the work of someone who did.

The 10 clusters are the stable ceiling. The question was always: what lives below?

TLCTC v2.0 formalizes a two-layer notation system: a **strategic layer** (`#X`) for human communication and a **operational layer** (`TLCTC-XX.YY`) for machine integration. The top-level cluster `TLCTC-XX.00` is reserved — it *must not* be used for any meaning other than the cluster itself. Sub-clusters at `YY ≠ 00` may express operational refinements without changing the top-level semantics.

What was missing was a **hierarchical convention** that makes the sub-cluster address space structurally readable — not a flat counter, but a tree.

The Convention: Tens Digit as Sub-Cluster, Ones Digit as Refinement

The proposal is straightforward. Instead of treating the two-digit suffix as a flat enumeration (`.01`, `.02`, `.03`…), we use the **tens digit** to identify the sub-cluster (the vector class) and the **ones digit** to identify refinements within that sub-cluster:

```
TLCTC-XX.00   ← top-level cluster (reserved, immutable)
```

```
TLCTC-XX.00   ← top-level cluster (reserved, immutable)
TLCTC-XX.10   ← sub-cluster 1 (vector class)
TLCTC-XX.11   ← refinement 1 within sub-cluster 1
TLCTC-XX.12   ← refinement 2 within sub-cluster 1
TLCTC-XX.20   ← sub-cluster 2 (vector class)
TLCTC-XX.21   ← refinement 1 within sub-cluster 2
 ...
```

This gives 9 sub-cluster slots (`.10` through `.90`) with 9 refinements each (`.X1` through `.X9`) — 81 operational positions per cluster. For the strategic layer, the corresponding shorthand is `#X.Y` (e.g., `#8.1` maps to `TLCTC-08.10`).

> **DESIGN PRINCIPLE**
>
> Every sub-cluster must answer the question: *"Through which vector does the attacker reach the same generic vulnerability?"* If the answer requires a different generic vulnerability, it belongs in a different cluster. If it's the same vulnerability reached through a different architectural path or physical mechanism — that's a legitimate sub-cluster.

To demonstrate that the architecture works, I refined four of the ten clusters — chosen because they exhibit the clearest natural vector decompositions. These refinements appeared first in the TLCTC v1.9.1 whitepaper and are here restated in the hierarchical notation convention.

#2 Exploiting Server

Generic vulnerability: *Code imperfection in server-side software.* The three vectors decompose *where in the server's software architecture* the imperfection resides.

```
TLCTC-02.00   #2   Exploiting Server (top-level, reserved)
TLCTC-02.10   #2.1   Protocol vector — server-side protocol handling flaws
TLCTC-02.20   #2.2   Core function vector — internal processing / parsing flaws
TLCTC-02.30   #2.3   External handler vector — delegated processing flaws
```

Examples: A Heartbleed exploit targeting the server's TLS implementation is `#2.1`. An SQL injection through the application's query parser is `#2.2`. A vulnerability in a

SQL injection through the application's query parser is #2.2. A vulnerability in a server-side PHP engine or Apache module is #2.3. The generic vulnerability is the same in all three cases — code imperfection on the server side — but the *vector* tells the defender where to look.

#3 Exploiting Client

Generic vulnerability: *Code imperfection in client-side software.* The same three-vector structure mirrors #2 — and this is not a coincidence. It falls directly out of Axiom VII (client-server as the fundamental interaction model). The code imperfection is the same vulnerability class; the three vectors decompose the same architectural layers, just on the other side of the interaction.

```
TLCTC-03.00   #3   Exploiting Client (top-level, reserved)
TLCTC-03.10   #3.1   Protocol vector — client-side protocol handling flaws
TLCTC-03.20   #3.2   Core function vector — internal processing / parsing flaws
TLCTC-03.30   #3.3   External handler vector — delegated processing flaws
```

The structural symmetry between #2 and #3 is analytically significant. It produces a 2×3 matrix of exploit vectors — server/client × protocol/core/handler — that is complete by construction. Any code exploit on any networked software component maps to exactly one cell.

#8 Physical Attack

Generic vulnerability: *Physical accessibility of IT assets.* The v1.9.1 whitepaper originally split this into "direct" and "indirect" sub-clusters. This update replaces those labels with **mechanical** and **signal** — a categorization based on the *physical mechanism of interaction* rather than the vague spatial concept of proximity.

```
TLCTC-08.00   #8   Physical Attack (top-level, reserved)
TLCTC-08.10   #8.1   Mechanical vector — physical contact with matter
TLCTC-08.20   #8.2   Signal vector — energy propagation, no contact required
```

Mechanical means the attacker physically touches or manipulates hardware: tampering, theft, intrusion into secure areas. Signal means the attacker exploits energy emissions or environmental conditions without direct contact: electromagnetic side-channels (TEMPEST), acoustic attacks, environmental manipulation. The

distinction is falsifiable — it's physics, not interpretation — and it maps to categorically different control regimes (physical access controls vs. signal shielding and emission standards).

#10 Supply Chain Attack

Generic vulnerability: *Necessary trust in third-party components, services, and processes.* The three vectors decompose the *temporal phase and medium* through which the trust relationship is exploited.

```
TLCTC-10.00   #10   Supply Chain Attack (top-level, reserved)
TLCTC-10.10   #10.1   Update vector — post-deployment, active delivery channel
TLCTC-10.20   #10.2   Development vector — pre-deployment, silent insertion
TLCTC-10.30   #10.3   Hardware vector — physical component supply chain
```

A compromised software update pushed to customers is `#10.1`. A backdoor injected into a build pipeline or malicious library dependency is `#10.2`. A hardware implant introduced during manufacturing is `#10.3`. The SolarWinds 2020 incident, for example, passes through `#10.1` at the trust acceptance event — the moment the organization's infrastructure accepts the compromised update because it trusts the vendor's distribution channel.

I deliberately stopped at four. The refinement of #1 (Abuse of Functions), #4 (Identity Theft), #5 (Man in the Middle), #6 (Flooding Attack), #7 (Malware), and #9 (Social Engineering) into sub-cluster vectors is analytically feasible — each has a generic vulnerability that can be decomposed by vector — but completing the work for all ten clusters requires resources, empirical validation, and community review that exceed what a single practitioner can responsibly deliver.

#1	#4	#5
Abuse of Functions	Identity Theft	Man in the Middle
VECTORS: OPEN	VECTORS: OPEN	VECTORS: OPEN

#6	#7	#9
Flooding Attack	Malware	Social Engineering
VECTORS: OPEN	VECTORS: OPEN	VECTORS: OPEN

The four completed refinements demonstrate the *method*. The architecture — hierarchical notation, generic vulnerability preservation, falsifiable vector distinction — is ready. The remaining six are not my limitation to apologize for. They are the work that institutions with operational data, research capacity, and standardization mandates are far better positioned to complete.

The sub-cluster architecture resolves a tension that has existed since TLCTC v1.0: the framework is strategically stable (10 clusters, cause-based, non-overlapping) but operationally it needed a clear path to technique-level granularity without compromising the top-level axioms.

The hierarchical notation provides that path. Attack path notation can now express operational precision where it matters — `#9 → #3.3 → #7` tells you not just "client exploit" but "external handler vector" — while remaining readable at the strategic level for anyone who only needs `#9 → #3 → #7`. The two layers are fully compatible. Nothing breaks.

For SIEM rules, the operational notation `TLCTC-03.30` enables automated correlation by vector class. For board reporting, `#3` remains sufficient. For threat intelligence sharing, the JSON architecture already supports sub-cluster precision at the step level. The infrastructure exists.

To NIST, MITRE, CISA, ENISA, and BSI

NIST CSF 2.0 defines ID.RA — Risk Assessment — but provides no standardized threat taxonomy. MITRE ATT&CK catalogs over 200 techniques but has no stable strategic abstraction layer. ENISA's annual threat landscape categories change year to year. CISA publishes advisories with no consistent

categories change year to year. CISA publishes advisories with no consistent causal classification. BSI's IT-Grundschutz mixes threats, vulnerabilities, and controls in a single catalog.

One person, working independently, derived a cause-based taxonomy of 10 clusters through systematic decomposition, demonstrated the refinement architecture with 4 proof-of-concept sub-cluster structures, formalized a hierarchical notation convention with room for 81 operational positions per cluster, and released everything under CC BY 4.0.

The remaining six clusters — and the empirical validation of all ten — need institutional capacity, operational datasets, and community review processes that standards bodies exist to provide. The method is documented. The notation is stable. The architecture is ready.

The question is no longer whether a unified, cause-based cyber threat taxonomy is possible. The question is whether the organizations with the mandate and resources to standardize it will do so — or whether the industry will continue to build control frameworks on a foundation that skips threat identification entirely.

> *I built the roof and four of the rooms. The blueprint is open. The tools are on the table. It would be unusual for institutions whose mandate is standardization to argue they need a larger house — while standing in one that has no foundation at all.*

References & Notes

1. Framework documentation, tools, and resources: tlctc.net
2. The TLCTC whitepaper v1.9.1 contains the original refinement discussion referenced in this article.

TLCTC Framework