So, what even is Threat Modelling? A security practitioner's guide

Dr Liz James

Contents

1. Introduction

Let's be honest: asking "what is threat modelling?" in a room full of security professionals is a great way to start an argument.

Ask someone from Governance, Risk, and Compliance (GRC), and they'll hand you a spreadsheet mapping STRIDE to a compliance framework. Ask a Red Teamer, and they'll show you a graph database of active directory attack paths. Ask an infrastructure engineer, and they'll start drawing trust boundaries and VPCs on a whiteboard.

One of the biggest points of friction in security teams is the monolithic treatment of this term. When a GRC analyst asks a developer for a "threat model" during a live operational phase using a template meant for the initial design phase, the result is inevitably frustration and a useless artifact.

Threat modelling is not a single, unified methodology. It is a cognitive framework whose inputs, outputs, and focus shift entirely depending on the objective and the stage of the system's lifecycle. To get any actual value out of the exercise, we need to decompose it.

Before we dive in, a bit of context on my own lens. I'm an engineer by training. I entered cybersecurity as a tester, got heavily exposed to the technical assurance side, and eventually moved into Consulting and Implementation. In that space, technical findings are just as important, but they have to sit within the wider organisation's risk appetite and frameworks.

Because of this journey, while I recognise that compliance is an essential tool for improving baselines, I have plenty of experience with fully compliant systems that are fundamentally insecure. I'm also not naive enough to assume there is such a thing as a perfectly 'secure' product, system, or process. Instead, we aim for "secure enough" (as secure as reasonably practicable given the organisation's appetite). More often than not, I find myself framing and reporting technical vulnerabilities not as pure technical issues, but as misalignments to that risk appetite.

With that pragmatic bias stated, here is how I've found threat modelling to break down in practice.

1. The 5 Lenses of Threat Modelling

Think of a complex environment like a multifaceted gem. Crucially, this metaphor applies equally whether you are dealing with traditional enterprise IT (cloud software, corporate networks) or highly Cyber-Physical Systems (Operational Technology, connected vehicles, medical devices, and industrial control systems).

The system itself doesn't change, but depending on the lens you hold up to it, entirely different features snap into focus while others blur into the background. A "lens" in this context is a combination of your objective, your lifecycle stage, and your overarching risk appetite.

In a traditional IT environment, looking through one of these lenses might highlight risks around data exfiltration or service downtime. Point that exact same lens at a Cyber-Physical System, and the focus immediately shifts to kinetic impact and human safety.

Regardless of the domain, each lens is designed to filter out irrelevant noise so you can answer a highly specific question. If you try to look through all of them at once, you'll just get a headache. If you look through the wrong one, you'll get a distorted view of reality that leads to terrible architectural or business decisions.

To actually get value out of threat modelling, you need to deliberately swap these lenses based on the problem you are trying to solve. Here are the five dominant lenses used across the industry today:

- Lens A: Governance, Regulation, and Compliance (GRC)
- Lens B: Security Engineering (The Root Cause Approach)
- Lens C: Attack Path Mapping (The Attacker / SecOps View)
- Lens D: Detection Engineering

- Lens E: Product, Business Logic, and Malicious Use of the Intended Function (MUIF)

Listing them out is easy but applying them is where the friction happens. To make this actionable, and to stop the cross-team arguments, we need to tear down the anatomy of each approach.

For each lens, we must define its core objective, what it strictly focuses on, where it fits into the system lifecycle, and the tangible artifacts it produces. Let's look at what happens when you snap each of these into focus.

1.1. Lens A: Governance, Regulation, and Compliance (GRC)

The Objective: Compliance, defensibility, and satisfying external or internal policy requirements.

The Focus: Completeness and categorization. This lens is heavily indexed on mapping an environment to established frameworks (like NIST CSF, ISO 27001, IEC 62443, or CIS Controls). GRC threat modelling wants to ensure that every system has been assessed, that known issues are categorized, and that any residual risk is formally documented and accepted by the business.

The Lifecycle Stage: Operational & Periodic. This is usually driven by the calendar rather than code commits think annual audits, vendor security renewals, major regulatory checkpoints, or post-merger integrations. It treats the system as a deployed, operating entity that needs to be governed.

The Output: Risk registers, compliance artifacts, system security plans (SSPs), and a verifiable paper trail demonstrating that security was thoroughly “considered”.

The Vibe: Top-down, methodical, and heavily structured.

The Reality Check: To an engineer, this lens often feels like bureaucratic box-ticking. But having sat in the consulting and implementation space for years, I can tell you this lens is critical for board-level conversations. However, it comes with a major caveat: compliance does not equal security. A system can perfectly align with an organisation’s risk appetite on paper, tick every ISO 27001 box, and still be trivially exploitable because the technical implementation is flawed. This lens is fantastic for raising the baseline and forcing accountability, but it relies entirely on the assumption that the underlying technical assessments were accurate in the first place.

1.2. Lens B: Security Engineering (The Root Cause Approach)

The Objective: Building resilient architecture from the bottom up and extinguishing entire classes of threat by design, rather than just patching symptoms.

The Focus: Component interactions, trust boundaries, and structural mechanics. This is where you look at the blueprints and ask, “If we enforce mutual TLS between these two microservices, do we completely render this entire category of MITM and spoofing threats irrelevant?” It’s about systemic fixes over point-in-time patches.

The Lifecycle Stage: Pre-deployment & Continuous Evolution. While this lens heavily champions “shift-left” (whiteboarding, secure design reviews, and threat modelling as code), it has a massive operational mandate. When a major vulnerability is found in production, the engineering model doesn't just ask “how quickly can we apply the vendor patch?” It asks, “how do we re-architect this segment to isolate the blast radius?” or “how do we retrofit identity-aware proxies so this legacy app isn't exposed?”.

The Output: Actionable design changes, secure defaults, mitigating controls as code, and “paved roads” that make the secure way the easiest way for developers to build.

The Vibe: Proactive, deeply technical, and builder-centric.

The Reality Check: As an engineer by training, this is naturally my favourite lens because it actually fixes the underlying problem. However, my consulting experience has taught me that this lens carries a major trap: engineering tunnel vision. Engineers naturally want to build the perfect, impenetrable fortress. But remember, there is no perfectly secure system, only “secure enough”. If an architectural overhaul costs a million pounds to mitigate a risk the business values at with an unlikely worst-case impact of £50,000, the engineering model has failed within the business context. We use this lens not to build perfect systems, but to structurally align the architecture with the organisation's accepted risk appetite.

1.3. Lens C: Attack Path Mapping (The Attacker / SecOps View)

The Objective: Understanding reality and prioritizing remediation based purely on the path of least resistance.

The Focus: Threat Intelligence (CTI), the kill chain, attacker economics, and graph theory. This lens completely ignores the neat, theoretical trust boundaries drawn in Lens B. It doesn't matter if a developer meticulously mitigated a theoretical CSRF vulnerability if an attacker can simply phish a session token, pivot through an unpatched VPN, and dump the production database.

The Lifecycle Stage: Operational & Assessment. This lens assumes the system is already built, deployed, and inevitably flawed. It is heavily utilized during continuous attack surface management, red team engagements, and post-deployment penetration testing.

The Output: Choke-point identification, exploit graphs, and highly prioritized remediation plans (focusing on the single step that breaks the kill chain, rather than blindly patching every high-CVSS finding).

The Vibe: Pragmatic, adversarial, and heavily exploit-driven.

The Reality Check: Coming from a testing background, this is the lens where theoretical security meets operational reality. In my early days as a tester, it became immediately obvious that attackers don’t think in lists; they think in graphs. When I consult today, I often see organisations drowning in vulnerability reports, desperately trying to patch thousands of isolated issues. This lens is the antidote. It helps us report technical vulnerabilities through their actual misalignment to the organisation's risk appetite. A CVSS 9.8 on an isolated, air-gapped test server is less dangerous than a CVSS 4.5 that happens to sit on the only viable pivot path to your Domain Controller. This lens forces you to fix the choke points that actually matter.

1.4. Lens D: Detection Engineering

The Objective: Visibility, rapid response, and acting as the active safety net for accepted risks or architectural failures.

The Focus: Post-compromise telemetry, logging gaps, and accepted risk. If Security Engineering (Lens B) determines a fix is too expensive, Detection Engineering steps in. The threat model here shifts completely away from prevention and asks: “If an attacker exploits this specific feature, what telemetry is generated? Do we actually have the logs? Can we alert on it before they pivot and achieve their final objective?”

The Lifecycle Stage: Operational & Continuous Tuning. This lens activates once the system is live. It is continuously refined based on incident post-mortems, the outputs of Attack Path Mapping (Lens C), and shifting threat intelligence.

The Output: SIEM rules, targeted logging configurations, detection-as-code, and heavily contextualized incident response playbooks.

The Vibe: Reactive but highly prepared, data-driven, and permanently living in the “assume breach” mindset.

The Reality Check: In my consulting work, this is the lens where “accepted risk” goes to live. When a business decides an architectural fix is too expensive, GRC (Lens A) will document it as an accepted risk on a register. But all too often, nobody tells the Security Operations Centre (SOC). You cannot truly accept a risk if you are completely blind to its exploitation. This lens forces the uncomfortable but necessary question: “Okay, we aren't structurally fixing this flaw. But if an attacker abuses it tomorrow at 3 AM, will we even know?”

1.5. Lens E: Product, Business Logic, and MUIF

The Objective: Protecting the core value proposition of the business, preventing platform abuse, and ensuring features aren't weaponized.

The Focus: Malicious Use of Intended Function (MUIF), fraud, and logical flaws. Standard security tooling is entirely blind here because there is no traditional “vulnerability” to exploit. This is about asking: “Can an attacker use our legitimate 'password reset' function to spam target inboxes?” or “Can they game our API rate limits to scrape our proprietary dataset or manipulate a referral program?”

The Lifecycle Stage: Ideation & Operational Feedback. This lens straddles the extremes. It must happen during the initial product discovery phase to prevent building abuse-friendly features in the first place. But it is also heavily refined during the operational phase, driven by real-world fraud metrics and Trust & Safety feedback loops.

The Output: Abuse cases (the evil twin of user stories), behavioural rate limiting, Trust & Safety controls, and strict business logic validations (e.g., "Step 3 cannot be executed unless Step 2 completed successfully").

The Vibe: User-journey focused, highly contextual, and frequently blurring the lines between Security, Trust & Safety, and Product Management.

The Reality Check: In my consulting experience, this is the lens where traditional security teams fail the hardest. I've seen clients spend fortunes on state-of-the-art WAFs and endpoint protection, only to bleed revenue because a malicious actor figured out how to automate the legitimate checkout flow to monopolize high-demand stock via automation. A vulnerability scanner will never flag a feature that is working exactly as it was designed to work. To model this effectively, you have to stop thinking like a hacker looking for a buffer overflow, and start thinking like a scammer looking for a loophole. Or, in the world of Cyber-Physical systems, start thinking about the kinetic energy involved. The "secure enough" balance here is entirely about risk appetite: how much friction are you willing to put in front of a legitimate user to stop a malicious one?

The Synthesis: What binds them together?

If these five lenses produce completely different artifacts, operate at different stages of the lifecycle, and look for fundamentally different types of flaws, is it even fair to call them all "threat modelling"?

Yes. Because despite the drastically different outputs, there is a shared cognitive DNA that binds all five of these approaches together. No matter which lens you are looking through, you are executing the same three fundamental processes:

1. **Abstraction (Distilling Complexity)** You cannot threat model reality; it is too noisy. Every single lens requires you to strip away the irrelevant details of a complex system and create a simplified, workable representation. GRC abstracts the system into control statements. Engineering abstracts it into data flows and trust boundaries. Attack Path Mapping abstracts it into a graph of interconnected nodes. The skill of the practitioner lies in knowing *what* to abstract and what to leave in.
2. **The "What If" Engine (Adversarial Thinking)** Whether you are an auditor asking, "What if the primary backup fails?" or a Red Teamer asking, "What if I forge a Kerberos ticket?", every lens is powered by structured, hypothetical, adversarial thinking. It is the deliberate process of looking at how something was *designed* to work, and methodically figuring out how it can be forced to fail.
3. **Prioritisation (Answering the "So What?")** This is the most critical commonality. The goal of threat modelling is never to fix everything. The goal is to answer a single question: *"Out of all the bad things that could happen, what should we care about right now, and what are we going to do about it?"* Whether the output is a boardroom risk register, a Jira ticket for an architectural refactor, or a new SIEM alert, the end result is always about prioritising action to align the system with the organisation's accepted risk appetite.

2. Taxonomies: The Translation Layer

If the five lenses dictate *how* we look at a system, we also need to talk about the language we use to describe what we find.

A major reason threat modelling efforts fail in large organisations is a breakdown in taxonomy. If you hand an engineer a spreadsheet full of ISO 27001 control deficiencies, they won't know how to code a fix. If you hand an auditor a raw list of Common Weakness Enumerations (CWEs), they won't know how to map that to business risk.

To make threat modelling work across the business, you need to understand which taxonomy belongs to which lens, and how to translate between them:

- **The Compliance & Risk Dialect (Common Control Frameworks):** This is the language of **Lens A (GRC)**. It includes NIST CSF, ISO 27002, CIS Controls, and bespoke CCFs. These frameworks speak in broad,

programmatic requirements (e.g., *"AC-2: Account Management"*). They are excellent for identifying systemic gaps across an enterprise, but they are practically useless for providing technical remediation advice to a developer.

- **The Structural Dialect (CWEs & Design Patterns):** This is the language of **Lens B (Security Engineering)**. Common Weakness Enumerations (CWEs) describe the underlying, structural root cause of a flaw (e.g., *CWE-89: Improper Neutralization of Special Elements used in an SQL Command*). Alongside methodologies like STRIDE, this taxonomy allows engineers to categorise and extinguish systemic architectural issues before they ever become exploitable vulnerabilities.

- **The Adversarial Dialect (MITRE ATT&CK & CVEs):** This is the language of **Lens C (Attack Path)** and **Lens D (Detection)**. MITRE ATT&CK provides a standardised taxonomy of attacker behaviour (Tactics, Techniques, and Procedures), while CVEs identify specific, known failures in software. This language ignores how the system was built and focuses entirely on how it can be broken and monitored.

- **The Product / Abuse Dialect (Bespoke Abuse Cases & CAPEC):** This is the language of **Lens E (MUIF)**. While libraries like CAPEC (Common Attack Pattern Enumeration and Classification) attempt to define "Abuse of Functionality," the reality is that the taxonomy here is highly bespoke to the business. Because MUIF relies on weaponising legitimate features, it is described using the language of fraud metrics, Trust & Safety playbooks, and user stories flipped upside down.

A mature security function doesn't just pick one of these taxonomies. Instead, it builds the connective tissue between them.

When a Red Team (Lens C) identifies a path of least resistance using **MITRE ATT&CK**, the engineering team (Lens B) maps that behaviour back to the root structural **CWE** to build a systemic fix, while the GRC team (Lens A) maps the accepted residual risk to the relevant **NIST Control** for the board. The taxonomies act as the gears that allow the different lenses to turn together.

3. Market Trends: Collapsing the Lenses

If you look at the current vendor landscape, the overarching theme is a desperate attempt to collapse these distinct lenses into a single pane of glass. While a true "unified" real-time threat model remains elusive, a few major trends are reshaping the space:

- **Continuous Threat Exposure Management (CTEM):** The shift away from point-in-time scanning toward continuous, graph-based validation. This is the collision of **Lens C (Attack Path)** and **Lens A (GRC)**, attempting to answer auditor questions with live data proving critical assets are mathematically unreachable.

- **Threat Modelling as Code (TMAC):** Moving models out of static Visio diagrams and writing them in YAML or Python alongside infrastructure code. This is **Lens B (Security Engineering)** evolving. If a developer breaks a trust boundary in Terraform, the CI/CD pipeline catches the logic flaw and fails the build before it ever reaches production.

- **Generative AI in the Design Phase:** While LLMs still struggle with the deep structural nuances of Lens B, they are wildly effective for **Lens E (MUIF)**. Feeding an AI a product spec and asking how a malicious user could weaponise the workflow is an incredibly fast way to bootstrap abuse cases and logically test the intended function at scale.

What are we at NCC Group doing about this?

Well, the first part of working to solve this problem and bridge the disconnect starts with simply recognising the issue. You can't fix a fragmented approach to threat modelling if you don't acknowledge that the fragments exist for valid, distinct reasons.

As a security consultancy, NCC Group naturally provides many of the discrete product and service lines I've described above. We have specialists who spend their entire lives in the architectural weeds of Security

Engineering, GRC experts who map compliance frameworks in their sleep, and Red Teams who view every system as a graph of exploitable attack paths.

But throwing three different threat models from three different teams over the fence to a client doesn't reduce friction, it usually amplifies it.

Where we find a customer needs to apply multiple lenses, which is essentially every mature or complex organisation, we bring in strategic advisors and domain Subject Matter Experts (SMEs) to help navigate the outputs. Their role is to act as the critical translation layer between these distinct paradigms.

They are there to help map the raw, technical outputs of a Security Engineering model into the assurance language required by the board, or to ensure that the harsh realities discovered during Attack Path Mapping actually inform the Detection Engineering pipeline. We aren't trying to force a single, mythical “unified” threat model; we are providing the right experts to ensure the outputs from one lens actively enrich the others, rather than competing with them.

4. Conclusion: Read the Room

The biggest takeaway here isn't that one lens is inherently better than the others (even if I will always heavily index on engineering). The takeaway is that applying the *wrong* lens to a problem is worse than doing no threat modelling at all.

Handing an engineer a GRC compliance checklist when they are trying to architect a secure microservice environment will only generate resentment. Handing an auditor an Attack Path graph when they just need to check a compliance box will only generate confusion. And releasing a new feature with a flawless engineering architecture, but completely ignoring MUIF, will result in your platform being weaponised on day one.

Threat modelling is a highly contextual tool. To be effective, you need to know exactly what outcome you are optimising for, where you are in the system's lifecycle, and most importantly, which room you are standing in.