# Practical Comparison of Threat Modeling Tools and Approaches for Small Teams

## 1. Executive Summary

In today's fast-moving development environments, threat modeling is essential for building secure systems. However, small teams often struggle with choosing the right approach. Some methods are too heavy and time-consuming, while others may lack depth or traceability. This whitepaper explores three practical approaches to threat modeling, with real-world examples and trade-offs:

(1) Threagile + RTMP + Incremental modeling,

(2) IriusRisk, and

(3) traditional complete schemes like STRIDE + DFD.

The aim is to help small-to-medium-sized teams select the right combination of tools and methods that balance completeness, scalability, and developer engagement.

## 2. Introduction

Threat modeling helps teams identify and mitigate potential security risks early in the development lifecycle. It's not just about compliance but about building secure products proactively. Yet the method and toolset a team chooses has a big impact on outcomes.

### Many small teams face these questions:

- Where do we start?

- Can we model threats without delaying sprints?

- How do we update models when our system evolves?

This whitepaper explores the spectrum of threat modeling tools, from full-system models to agile-friendly approaches. We compare leading tools using concrete examples and introduce a hybrid approach that combines structured modeling (Threagile), fast prototyping (RTMP), and incremental threat modeling (ITM).

## 3. Threat Modeling Tools: Landscape Overview

Threat modeling tools can be broadly categorized based on two key dimensions:

1. **Licensing model** – *Commercial* vs. *Open Source*
2. **Modeling approach** – *With code*, *From code*, or *Via diagrams/forms*

Each category reflects how teams prefer to integrate threat modeling into their development and security processes — whether tightly coupled to code, aligned with diagrams, or supported by automation and rulesets.

### Commercial Tools

These tools are typically used in enterprise settings and focus on structured, repeatable threat modeling workflows. They offer rich features like compliance mapping (e.g., ISO, NIS2, OWASP), integrations with CI/CD pipelines, and dashboards for risk management.

- **IriusRisk**
  A mature, rules-driven platform that generates threat models using component questionnaires, architectural diagrams, and a vast library of attack patterns and countermeasures. Ideal for teams seeking automated coverage, compliance alignment, and integration into SDLC workflows.
- **ThreatModeler**, **Totamantic**
  These offer similar capabilities in terms of diagramming and automated threat generation, though with different UX models and pricing structures. They typically require more up-front investment and training but offer powerful enterprise features.

**Pros:** Automation, regulatory traceability, integration-ready
**Cons:** Licensing cost, steeper learning curve, less flexibility for ad hoc modeling

### Open Source Tools:

Open-source tools are often favored by small teams, researchers, or developers who want flexibility, scriptability, or free access. While they may not match the enterprise polish of commercial tools, many provide strong foundations for secure design workflows.

- **Threagile**
  YAML-based threat modeling where architecture and data flows are described as code. It auto-generates risk reports using over 40 built-in rules and supports custom risk categories, STRIDE mapping, and ASVS/CWE tagging.
- **PyTM**
  Python-based, object-oriented framework for defining systems and generating threats through embedded logic. Offers high flexibility and scripting power for DevSecOps engineers.
- **OWASP ThreatDragon**, **Draw.io**
  Diagram-driven tools that help visualize systems and manually annotate threats. Useful for initial workshops and visual models, though lacking automation.

- **Microsoft TMT**
  A classic tool for STRIDE + DFD modeling, with predefined threat libraries. No longer actively maintained but still used in legacy contexts.

**Pros:** Free, flexible, ideal for workshops or small teams
**Cons:** Limited automation, manual updates, lack of native CI/CD support

### Categories by Modeling Approach:

Understanding **how** a tool fits into your development process is as important as the tool itself. Threat modeling tools can be grouped by the modeling approach they support:

| Approach | Description | Tools |
|---|---|---|
| **Model with code** | System is described programmatically (e.g., YAML, Python) | Threagile, PyTM |
| **Model from code** | Tool attempts to extract model from source code annotations (less common) | ThreatSpec (deprecated) |
| **Diagram/form-driven** | Model built via visual tools or guided forms | IriusRisk, ThreatDragon, TMT |

Each approach serves different needs:

- **Model with code**: Best for dev-centric teams, automation, and version control
- **Diagram/form-driven**: Ideal for visual thinkers, workshops, or cross-functional reviews
- **Model from code**: Ambitious, but often fragile or abandoned due to complexity

## 4. Three Core Approaches Compared

### 4.1 IriusRisk: Complete, Enterprise-Grade Modeling

IriusRisk is designed for security teams and architects. It provides structured asset libraries, risk rules, and countermeasure suggestions. It integrates well with compliance needs but requires a complete system view upfront.

- Example (see Annex): In a case modeling an IoT fridge system with MQTT and cloud, IriusRisk generated detailed threats and mitigation plans across hardware, software, and communication layers.

#### Strengths:

- Deep threat coverage

- Built-in rules for cloud, APIs, containers

- Good for compliance workflows

**Weaknesses:**
- Requires up-front system clarity

- Less agile or feature-first friendly

## 4.2 Threagile + RTMP + Incremental: Agile & Dev-Friendly

This hybrid approach suits agile teams. RTMP (Rapid Threat Model Prototyping) offers fast, collaborative analysis. Threagile then codifies the architecture in YAML and auto-generates a full risk report. As new features are added, Incremental Threat Modeling (ITM) helps extend the model over time.

- Example (see Annex): A 3-tier web app with a "Share To-Do List" feature was modeled using this approach. RTMP revealed threats like token leakage, overprivileged sharing, and lack of logging. Threagile formalized these as risks, and added automated findings like XSS and missing cloud hardening.

**Strengths:**
- Works per feature or per sprint

- Low setup time, high automation potential

- Includes developer-driven insights

**Weaknesses:**
- Less built-in business context unless extended

- Requires some YAML fluency

## 4.3 Classic TM Scheme (DFD + STRIDE Full System)

This traditional method models the full system with Data Flow Diagrams and applies STRIDE across components and boundaries. It's often used in formal risk reviews.

**Strengths:**
- Completeness

- Formal traceability

**Weaknesses:**
- Time-intensive

- Difficult to maintain as systems evolve

- Often divorced from sprint cadence

## 5. Trade-Offs Table

| Criteria | Threagile + RTMP + Incremental | IriusRisk | Classic TM |
|---|---|---|---|
| **Setup Time** | Short | Medium | Long |
| **Detail Level** | Medium-high | High | Very High |
| **Ease of Use** | High | Medium | Low |
| **Team Size Fit** | Small–medium | Medium–large | Large, specialized |
| **Adaptability** | High | Medium | Low |
| **Output** | YAML + PDF + diagrams | Dashboard, reports | Docs + diagrams |

## 6. Real-World Observations

In practical settings, development teams frequently encounter challenges in applying threat modeling consistently. One common difficulty is determining **how much modeling is enough**—teams often ask whether they need to cover the full architecture or focus only on critical features. This uncertainty can lead to delays or incomplete models.

Success is more likely when teams adopt an **incremental approach**: starting small, focusing on high-impact components, and expanding the model over time. Rather than aiming for exhaustive system coverage from the start, teams benefit from integrating threat modeling into their existing development cadence—often during design reviews or sprint planning.

Lightweight techniques such as **Rapid Threat Model Prototyping (RTMP)** have proven especially useful. They allow teams to quickly reason about potential threats without requiring formal tools or deep security expertise. These collaborative sessions help shift the mindset from "security as a checklist" to "security as a design conversation," which accelerates both learning and adoption.

Ultimately, the most effective threat modeling efforts are those that balance structure with practicality—providing just enough rigor to uncover risks, while remaining adaptable to changing systems and delivery pressures.

## 7. Best Practice Guidance

To ensure that threat modeling becomes a sustainable and impactful part of secure software development, teams should follow a few proven practices:

- **Prioritize high-risk features first.** Begin modeling with components that are exposed to the internet, process sensitive data, or connect across trust boundaries. These areas are most likely to be exploited and often yield the most critical insights.
- **Integrate modeling into agile rituals.** Lightweight methods like Rapid Threat Model Prototyping (RTMP) can be run during sprint planning, architecture reviews, or feature refinement sessions. This ensures threat modeling remains actionable, timely, and collaborative.

- **Formalize models for reuse and automation.** Tools like Threagile enable teams to codify threat models in YAML, making them versionable, reviewable, and reusable. This allows threat modeling to scale as the architecture evolves.
- **Leverage structured tools for compliance alignment.** For teams working under formal standards such as ISO 27001, NIS2, or secure SDLC frameworks, platforms like IriusRisk provide built-in mappings to security controls and facilitate audit-ready documentation.

The most effective threat modeling practices are those that align with the team's delivery pace, maturity, and security goals—starting simple and evolving iteratively as confidence and complexity grow.

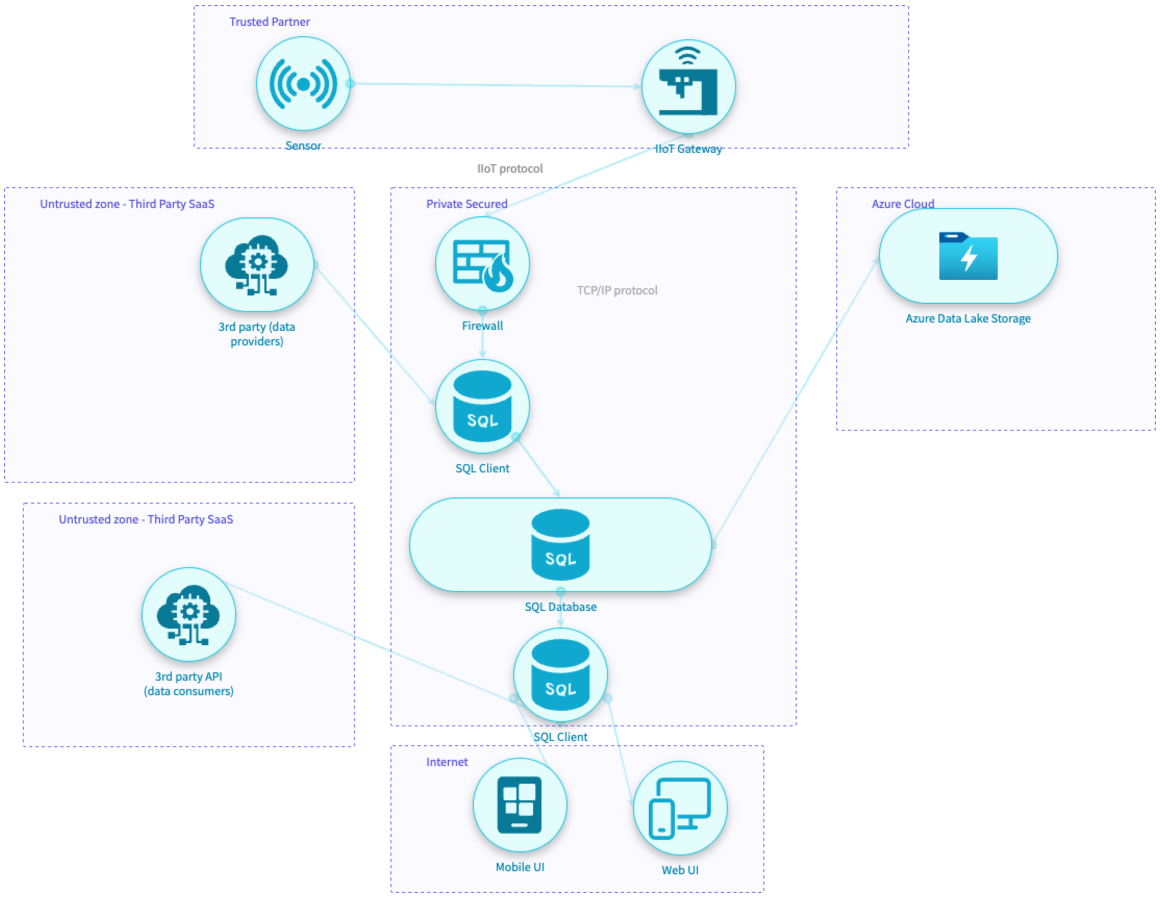
## 8. Conclusion

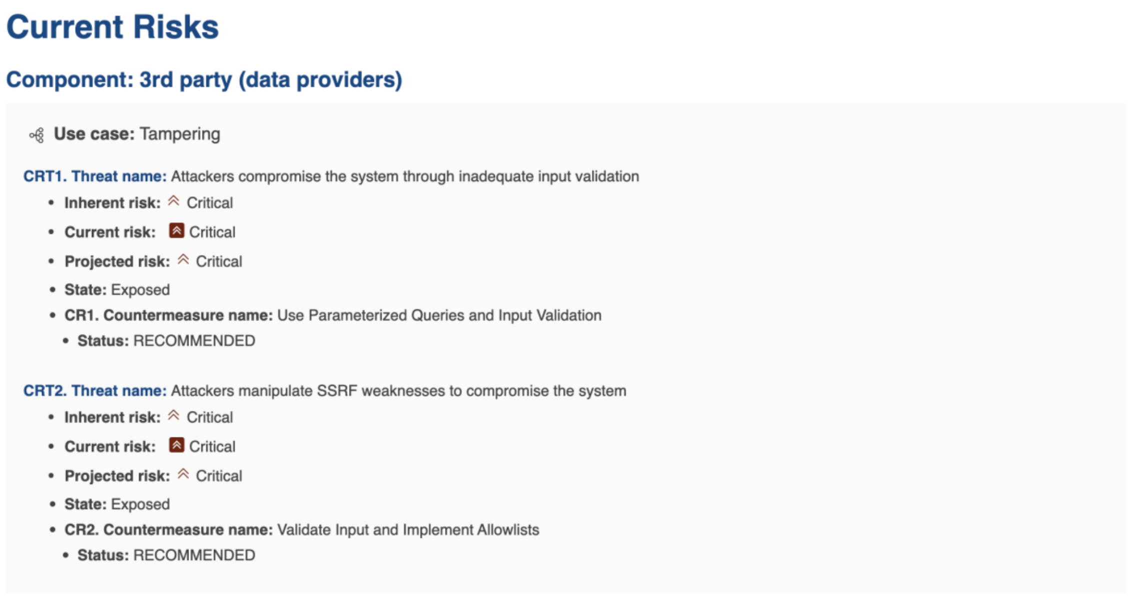
Threat modeling is not one-size-fits-all. The best tool is the one your team will actually use. For many small-to-medium dev teams, combining fast prototyping (RTMP), automated modeling (Threagile), and iterative growth (ITM) offers a practical path forward. Larger teams or those under formal regulation may benefit from tools like IriusRisk.

In all cases, treat threat modeling as a living process, not a one-time deliverable.

## 9. Appendices

## Appendix A: IriusRisk Threat Modeling example





### Current Risks

#### Component: 3rd party (data providers)

**Use case:** Tampering

**CRT1. Threat name:** Attackers compromise the system through inadequate input validation

- **Inherent risk:** Critical
- **Current risk:** Critical
- **Projected risk:** Critical
- **State:** Exposed
- **CR1. Countermeasure name:** Use Parameterized Queries and Input Validation
  - **Status:** RECOMMENDED

**CRT2. Threat name:** Attackers manipulate SSRF weaknesses to compromise the system

- **Inherent risk:** Critical
- **Current risk:** Critical
- **Projected risk:** Critical
- **State:** Exposed
- **CR2. Countermeasure name:** Validate Input and Implement Allowlists
  - **Status:** RECOMMENDED

(these are critical risks – a sample of report. See full report in external document)

## Appendix B: RTMP Method Overview

**Rapid Threat Model Prototyping (RTMP)** is a lightweight, developer-friendly approach to threat modeling that prioritizes **speed, structure, and repeatability**. Created by Geoff Hill, RTMP emerged as a response to the limitations of traditional threat modeling methods in agile and CI/CD environments, where long workshops and full-system modeling are often impractical.

### Core Principles

- **Intentional Architecture**: RTMP builds on existing diagrams or high-level architecture models rather than requiring formal DFDs. It leverages what developers already have.
- **Zones of Trust**: The method uses **numbered trust zones** to quickly identify boundaries and assess flows. Trust zone math is used to surface common STRIDE threats (e.g., EoP when moving from lower to higher zones).
- **Just-Enough Information**: RTMP applies the **80/20 rule**—capture enough threat data to take action, but don't aim for exhaustive coverage in the first pass.
- **Repeatability & Automation**: Outputs are consistent, measurable, and automatable. The method is designed to plug into CI/CD workflows via structured formats (JSON, CSV).

### Process Overview

1. **Start from Existing Context Diagram** (or draw a simple one)
2. **Identify Trust Zones** and assign numerical values (e.g., 0 = internet, 1 = app, 2 = DB)
3. **Apply Zone Math** to detect STRIDE threats:
    - Upward flow → Elevation of Privilege
    - Downward flow → Info Disclosure
    - Cross-zone flows → Tampering, Spoofing, Repudiation
4. **Estimate CIA (Confidentiality, Integrity, Availability)** for each flow
5. **Score risks** using impact heuristics (T-shirt size or numeric)
6. **Map mitigations** based on STRIDE category (e.g., TLS for Info Disclosure)
7. **Export results** into tools like Jira or generate security testing guidance

### Benefits for Agile Teams

- Fast to start and easy to teach in workshops
- Enables **"model storming"**—collaborative, ad-hoc analysis as features evolve
- Works well in **incremental modeling**: model new features during sprint planning
- Allows teams to **build a baseline model**, then improve coverage over time

RTMP is ideal for teams who want to embed threat modeling into development without heavy upfront investment. It can also be combined with tools like **Threagile** to formalize outputs after collaborative sessions.

## Appendix C: Sample Threagile YAML

```
threagile_version: 1.0.0
title: "To-Do App \u2013 Share Feature"
date: '2025-03-25'
author:
  name: Small Dev Team
  homepage: https://example.com
business_criticality: important
management_summary_comment: Threat model for Share To-Do List feature in a 3-tier
  web app.
business_overview:
  description: Simple web-based to-do list app with sharing functionality.
technical_overview:
  description: React frontend, Node.js API, PostgreSQL DB, email-based sharing.
questions:
  How are the admin clients managed/protected against compromise?: ''
  How are the development clients managed/protected against compromise?: ''
  How are the build pipeline components managed/protected against compromise?: ''
tags_available:
- react
- nodejs
- postgresql
- sendgrid
data_assets:
  todo-items:
    id: todo-items
    description: User to-do list items
    usage: business
    origin: User
    owner: Small Dev Team
    quantity: many
    confidentiality: confidential
    integrity: critical
    availability: important
    justification_cia_rating: Users expect confidentiality, integrity, and moderate
      availability for daily planning.
  share-metadata:
    id: share-metadata
    description: Sharing metadata including tokens and expiration
    usage: business
    origin: App
    owner: Small Dev Team
    quantity: many
    confidentiality: confidential
    integrity: critical
    availability: important
    justification_cia_rating: Controls access to shared lists; must be tamper-proof
      and moderately available.
  email-content:
    id: email-content
    description: Email body containing tokenized sharing link
    usage: business
    origin: App
    owner: Small Dev Team
    quantity: many
    confidentiality: confidential
    integrity: critical
    availability: operational
    justification_cia_rating: Includes links that allow access to to-do lists.
technical_assets:
  frontend:
    id: frontend
    description: React frontend used by users
    type: external-entity
    usage: business
    used_as_client_by_human: true
    out_of_scope: false
    size: application
    technology: browser
    internet: true
    machine: physical
    encryption: none
    owner: User
    confidentiality: confidential
    integrity: critical
    availability: important
    justification_cia_rating: Frontend needs to display sensitive to-do data securely.
    multi_tenant: false
    redundant: false
    custom_developed_parts: true
    data_assets_processed:
    - todo-items
  share-api:
    id: share-api
    description: Node.js API handling the share feature
    type: process
    usage: business
    used_as_client_by_human: false
    out_of_scope: false
    size: service
    technology: web-application
    internet: false
    machine: container
    encryption: transparent
    owner: Small Dev Team
    confidentiality: confidential
    integrity: critical
    availability: critical
    justification_cia_rating: Handles access control and sharing logic.
    multi_tenant: false
    redundant: false
    custom_developed_parts: true
    data_assets_processed:
    - todo-items
    - share-metadata
    - email-content
    data_assets_stored:
    - share-metadata
  mailer-service:
    id: mailer-service
    description: External email service
    type: external-entity
    usage: business
    used_as_client_by_human: false
    out_of_scope: true
    justification_out_of_scope: External email sending service (e.g. SendGrid)
    size: component
    technology: web-service-rest
    internet: true
    machine: serverless
    encryption: transparent
    owner: SendGrid
    confidentiality: confidential
    integrity: critical
    availability: important
    justification_cia_rating: Handles sensitive outbound emails with tokens.
    multi_tenant: true
    redundant: true
    custom_developed_parts: false
    data_assets_processed:
    - email-content
  postgres-db:
    id: postgres-db
    description: PostgreSQL database storing to-do items and shares
    type: datastore
    usage: business
    used_as_client_by_human: false
    out_of_scope: false
    size: component
    technology: database
    internet: false
    machine: virtual
    encryption: data-with-symmetric-shared-key
    owner: Small Dev Team
    confidentiality: confidential
    integrity: critical
    availability: critical
    justification_cia_rating: Holds all user content and sharing state.
    multi_tenant: false
    redundant: true
    custom_developed_parts: false
    data_assets_stored:
    - todo-items
    - share-metadata
trust_boundaries:
  internet-zone:
    id: internet-zone
    description: Untrusted user internet space
    type: network-cloud-security-group
    technical_assets_inside:
    - frontend
  app-zone:
    id: app-zone
    description: Application runtime network
    type: network-cloud-security-group
    technical_assets_inside:
    - share-api
  db-zone:
    id: db-zone
    description: Internal DB network
    type: network-cloud-security-group
    technical_assets_inside:
    - postgres-db
```

## Appendix D: STRIDE Reference Table

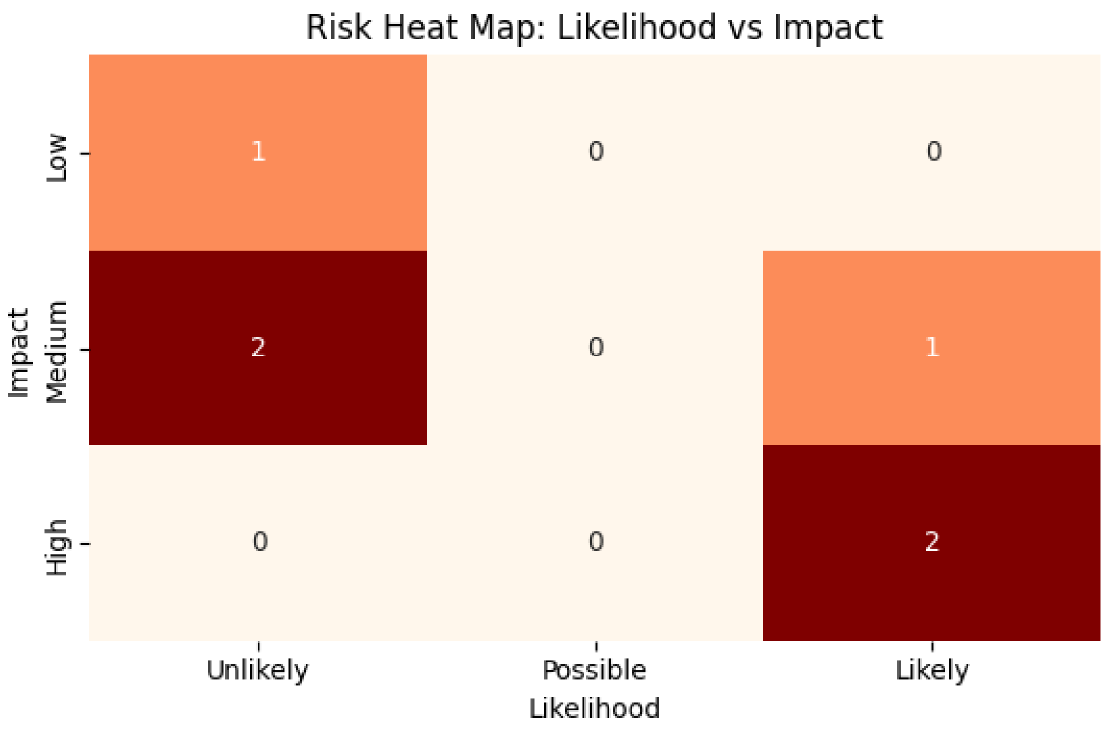
| Threat Type | Security Property at Risk | Definition | Examples | Standard Mitigations |
|---|---|---|---|---|
| **Spoofing** | Authentication | Impersonating something or someone else. | Pretending to be a user or process (e.g., faking credentials, spoofing an IP address). | - Cookie/Kerberos/NTLM authentication- SSL/TLS with certificates- PKI, IPSec- Digital signatures, MACs, hashes |
| **Tampering** | Integrity | Unauthorized modification of data or code. | Altering configuration files, modifying packets in transit, or injecting code. | - Access control lists (ACLs)- Digital signatures- Message authentication codes- File integrity monitoring |
| **Repudiation** | Non-repudiation | Claiming to have not performed an action. | Denying having sent a message or accessed a resource. | - Secure logging and auditing- Digital signatures- Trusted time stamps- Strong user identity verification |
| **Information Disclosure** | Confidentiality | Exposing information to unauthorized parties. | Data leaks, verbose error messages, unsecured APIs exposing sensitive data. | - Encryption (at rest & in transit)- ACLs and need-to-know access- Secure defaults and data classification |
| **Denial of Service** | Availability | Making a system or resource unavailable to legitimate users. | Flooding a server with requests, crashing an app, CPU/memory exhaustion. | - Input filtering and rate limiting- Quotas- High availability designs- Resource isolation- Authorization controls |
| **Elevation of Privilege** | Authorization | Gaining higher privileges than intended. | Regular user executing admin-level actions, container breakout. | - Principle of least privilege- Role-based access control (RBAC)- Input validation- Privilege separation- Secure configuration of system permissions |

### Notes:

- Each STRIDE threat maps to one of the **CIA+AA** security properties (Confidentiality, Integrity, Availability, Authentication, Authorization, Non-repudiation).

- **Standard mitigations** should be chosen based on context, and multiple controls may be layered for defense-in-depth.
- These mitigations are aligned with Microsoft SDL and industry practices

## Appendix E: Risk Comparison Heatmap



The heat map provides a visual summary of risk distribution based on two key dimensions:

- **Likelihood** – how probable it is that the threat will materialize
- **Impact** – the potential damage if the threat is realized

### Key Observations

- **High Impact, Likely Risks (Top-Right Corner):**
  - This quadrant contains the most critical threats. In this example, both **token reuse/prediction** and **unencrypted data flow** fall here.
  - These are *must-address* risks — they affect confidentiality or integrity and are easily exploitable in real-world scenarios.
- **Medium Impact, Likely Risks:**
  - Risks like **cross-site scripting (XSS)** fall into this zone. While the damage may be more limited, their high likelihood still warrants mitigation — especially in user-facing web applications.
- **Medium Impact, Unlikely Risks:**

    - Items such as **missing vaults for secrets** fall here. They may not be immediately exploitable but represent weak infrastructure hygiene and could be chained in an attack path.
- **Low Impact, Unlikely Risks (Bottom-Left Corner):**
    - **Unnecessary technical assets** are in this zone. These are typically *low-priority clean-up items* that might reduce complexity or future attack surface but don't demand urgent attention.

### Strategic Takeaways

- The heat map clearly identifies **prioritization tiers**:
    - **Top-right risks** should be addressed in the next sprint or release.
    - **Middle-tier risks** deserve tracking and planning for mitigation over time.
    - **Lower-tier risks** can be documented and deferred unless they become part of a broader chain.
- By plotting risks this way, teams can focus their limited security resources on what matters most — improving both effectiveness and decision-making transparency.

## Appendix F: Resources and Links to Tools

Irius Risk community edition: https://community.iriusrisk.com/

Microsoft Threat Modeling Tool: https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool

Threagile: https://threagile.io/

Threat Modeling by Izar Tarandach (A practical Guide for Development Teams)

Rapid Threat Modeling Prototyping: https://github.com/geoffrey-hill-tutamantic/rapid-threat-model-prototyping-docs

Incremental Threat Modeling:
https://2017.appsec.eu/presos/CISO/Incremental%20Threat%20Modelling%20-%20Irene%20Michlin%20-%20OWASP_AppSec-Eu_2017.pdf

OWASP Threat Modeling: https://owasp.org/www-community/Threat_Modeling