Using Combinatorial Testing to extend STRIDE

Nika Jeršič[1,*], Muhamed Turkanović[1]

[1]University of Maribor, The Faculty of Electrical Engineering and Computer Science, Koroška cesta 46, 2000 Maribor, Slovenia

Abstract

Threat modeling is a structured, design-time process for understanding a system, identifying what can go wrong, and deciding how to mitigate security risks before deployment. Although threat modeling is commonly recommended early and be maintained throughout the software development life cycle (SDLC), practical adoption remains uneven, especially in large environments. This paper presents STRIDEX, an approach that uses combinatorial testing strategies to extend STRIDE threat modeling framework. The paper describes systematic construction of multi-category attack scenarios by applying five combinatorial algorithms - Cartesian Product, Pairwise Testing, Equivalence Partitioning, Attack Variations, and Permutations. This work focuses specifically on scenario generation and validation; the generated scenarios are enriched with offline attack vector knowledge to support analyst review. We present prototype implementation, evaluate the five algorithms with a multi-criteria scoring scheme, and discuss the practical implications of algorithm selection for threat modeling practitioners. The approach is intended to support deterministic threat analysis across systems with tens of services while preserving repeatability and enabling consistent, evidence-based risk assessment.

Keywords

Threat modeling, STRIDE, Combinatorial testing

1. Introduction

Information security threat modeling is a standard activity in the software development lifecycle. It allows development and security teams to systematically identify, enumerate, and mitigate potential threats before systems are deployed. Among the many threat modeling frameworks in use today, STRIDE remains one of the most widely taught and applied [1]. STRIDE classifies security threats into six categories that cover common classes of attacks against information systems. Spoofing (S) refers to a violation of authentication, where an attacker impersonates a user or process. Tampering (T) represents a breach of integrity, involving the unauthorized modification of data. Repudiation (R) violates non-repudiation by enabling an entity to deny that an action took place. Information Disclosure (I) compromises confidentiality by exposing data to unauthorized parties. Denial of Service (D) affects availability by degrading or disrupting a service. Elevation of Privilege (E) constitutes a violation of authorization, where an attacker gains permissions beyond those that are legitimately granted [2, 3].

Despite its conceptual clarity, applying STRIDE to realistic enterprise systems does not scale [4]. A single component may attract threats from all six categories, and a system with n components each having k active categories and v threats per category yields a threat space on the order of v^k per component-exponential in the number of categories. In a small system with three or four components, a skilled analyst can walk through each STRIDE category for each component in a reasonable time. The problems emerge at scale. Enterprise systems commonly have 20 to 50 discrete services, each potentially relevant to several STRIDE categories, each with multiple concrete threats per category. Working through all meaningful combinations manually is inconsistent because different analysts working on the same system will produce different, incomparable outputs [5]. Additionally, modern software systems evolve continuously. Each time a new service is added, a trust boundary shifts, or a data flow changes, the threat model needs updating [6].

SQAMIA 2026: Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications, September 16–18, 2026, Niš, Serbia

*Corresponding author.

nika.jersic@um.si (N. Jeršič); muhamed.turkanovic@um.si (M. Turkanović)

0009-0008-4644-878X (N. Jeršič); 0000-0002-5079-5468 (M. Turkanović)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-19

CEUR-WS.org/Vol-4336/paper9.pdf

From this perspective, the STRIDE analysis of large, evolving systems becomes a problem of combinatorial interactions: the challenge lies in exploring the high-dimensional space of combinations of categories and components without having to thoroughly list every possibility.

Combinatorial testing (CT) is a branch of software testing in which input parameters are systematically combined to maximize fault-detection coverage while constraining the number of test cases [7]. It is based on the principles of statistical design of experiments (DoE) and is used in various industries, particularly for high-risk software applications that affect safety or security [8]. The method is considered de facto exhaustive or pseudo-exhaustive, as it can achieve a similar level of defect detection as fully exhaustive testing while reducing the size of the test suite, often by a factor of 20 to more than 100 [8]. The importance of combinatorial testing stems from the ever-increasing complexity of software systems, which often leads to a phenomenon known as the "combinatorial explosion", where the number of possible combinations of input data becomes exponentially large, making exhaustive testing impractical [9].

To support enterprise-level threat analysis, we propose an extended STRIDE approach called STRIDEX, intended to assist information security specialists in producing consistent and repeatable outputs at scale. The core process in STRIDEX is combinatorial testing (CT): instead of evaluating each category in isolation, the method systematically combines threats across active STRIDE categories to generate multi-category attack scenarios with explicit interaction coverage. To operationalize this methodology in practice, we also provide the STRIDEX analyser tool, which implements the full pipeline from architecture input and STRIDE threat elicitation, through combinatorial scenario generation, to enrichment with real-world attack vectors for analyst review. The mapping between combinatorial testing concepts and STRIDEX threat analysis, shown in Table 1.

Table 1
Conceptual mapping from combinatorial testing to STRIDE analysis

Combinatorial testing concept	**STRIDEX**
Parameter	STRIDE category (S, T, R, I, D, E)
Parameter value	A concrete threat row from the input
Test case	A multi-category attack scenario
t-way coverage	All t-category combinations appear in at least one scenario
Defect	A real-world attack exploiting multiple STRIDE categories

Under this mapping, each STRIDE category becomes a dimension of the input space, each threat becomes a value in that dimension, and each generated scenario is a test case. The goal is to surface attack scenarios that require simultaneous exploitation of multiple STRIDE categories, which are often underrepresented in manual analysis.

Methodologically, this article follows problem identification, artifact design, demonstration and evaluation. First, STRIDE is mapped to combinatorial testing concepts in order to define how threats, categories, and scenarios correspond in the STRIDEX model. Second, a deterministic STRIDEX analyser prototype is designed and implemented to operationalise the elicitation-combination-enrichment pipeline described in the paper. Third, the paper compares five combinatorial strategies using a multi-criteria scoring scheme that considers coverage, scalability, reviewability, realism, and compliance fit. Together, these methods bring together the conceptual foundation, programmatic implementation, and practical trade-offs of this approach.

The study is guided by two research questions:

- RQ1: How do different combinatorial strategies change the volume in STRIDE scenario generation?
- RQ2: Which strategy profiles are most suitable for different analysis intents?

Following a design-science perspective, we treat STRIDEX as a prototype artifact that is implemented and demonstrated on representative profiles, and evaluated through a comparison of combinatorial

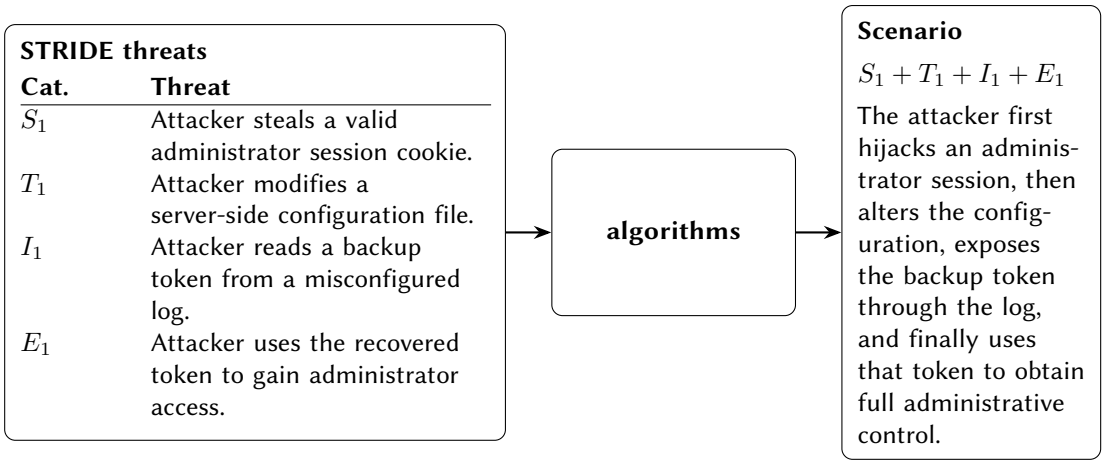


Figure 1: Illustrative principle of workflow from STRIDE to STRIDEX

strategies. Compared with conventional STRIDE, which produces isolated per-category threat lists, STRIDEX treats the system as a combinatorial problem in which cross-category attack paths are generated.

The remainder of the paper is organised as follows. Section 2 presents STRIDEX with key entities, involved roles, workflow and STRIDEX Analyser tool. Section 3 presents common combinatorial algorithms and presents a comparative summary of these algorithms. Section 4 concludes the paper.

2. STRIDEX: Conceptual and Process Model

This section presents STRIDEX conceptual model that unifies STRIDE with combinatorial testing. The section also presents what the formal inputs and outputs of the process are, and how the three conceptual processing layers interact to transform a system description into an enriched, actionable scenario set.

STRIDEX (STRIDE extended by cross-category combinatorial analysis) restructures the output space of threat analysis. Instead of a flat per-category threat register, STRIDEX produces a scenario register whose elements are multi-dimensional with each element containing threats from each active STRIDE category. The conceptual shift from STRIDE to STRIDEX is summarised in Table 2. An example is also presented in Figure 1 that illustrates the basic unit of analysis for the algorithms in this section: each method determines how such category-threat combinations are selected, ordered, or compressed to produce the final scenario set. The principle is the same for all algorithms.

Table 2
STRIDE and STRIDEX: conceptual comparison

Dimension	STRIDE	STRIDEX
Unit of analysis	Single-category threat	Multi-category attack scenario
Output structure	Per-category list	Scenario set
Coverage guarantee	None (analyst-dependent)	Formally defined by algorithm
Inter-category interactions	Implicit, often missed	Explicit, guaranteed by construction
Determinism	Varies by analyst	Identical inputs $\Rightarrow$ identical output
Enrichment	Manual research	Enriched by intelligence database

2.1. Key entities

The key entities represent the main functional building blocks of the model, each responsible for a specific stage in the model structure, from data input and threat elicitation to scenario generation and

intelligence enrichment.

- **System Model:** Structured architecture description (components, data flows, trust boundaries) used as the basis for threat elicitation.
- **Threat:** A specific security concern tied to a component and STRIDE category, including attack descriptions and mitigations.
- **Threat Profile:** The complete, organized collection of system threats (derived manually or automatically) that serves as input for scenario generation.
- **Attack Scenario:** A coordinated attack combining threats across multiple STRIDE categories (maximum one threat per category).
- **Attack Vector:** A severity-rated, real-world attack technique used to enrich generated scenarios.

2.2. Roles

- **System Architect:** Defines the system architecture (components, data flows, boundaries) and contextual metadata (exposure, authentication, data classification).
- **Security Analyst:** Validates threat profiles, selects combinatorial strategies, and evaluates results for risk prioritization.

2.3. Processes

1. **Input Data — Architecture Description**
   System components, data flows, trust boundaries, and metadata are normalized from formal models (e.g., UML, ArchiMate, DFD) or manual input.
2. **STRIDE Generation — Threat Classification**
   Architectural elements are mapped to the six STRIDE categories, creating a standardized threat profile.
3. **Scenario Generation — Combinatorial Analysis**
   STRIDEX combines threats across STRIDE categories into multi-vector scenarios using one of five algorithms: Cartesian product, pairwise testing, equivalence partitioning, attack variations, or permutations.
4. **Scenario Enrichment — Threat Intelligence Mapping**
   Scenarios are enriched with severity-rated techniques from an attack-vector database to support risk assessment and mitigation planning.

2.4. STRIDEX Analyser tool

This subsection presents the STRIDEX analyser prototype, a Python web application that implements the proposed methodology. It uses a FastAPI backend and relational database to manage analysis sessions across three layers: elicitation, combinatorial analysis, and intelligence enrichment.

As shown in Figure 3, system input is collected through the Infrastructure Builder, a five-step wizard (System Information, Trust Zones, Components, Flows, Groups), after which the analyst selects one of the five combinatorial strategies. This input is forwarded to the Elicitation layer, which generates a structured STRIDE analysis per component. The Combinatorial layer then applies the selected algorithm (Cartesian product, pairwise testing, equivalence partitioning, attack variations, or permutations) to produce a raw scenario set, which the Intelligence layer enriches by querying a local attack-vector database filtered by STRIDE category, element type, keyword, and severity.

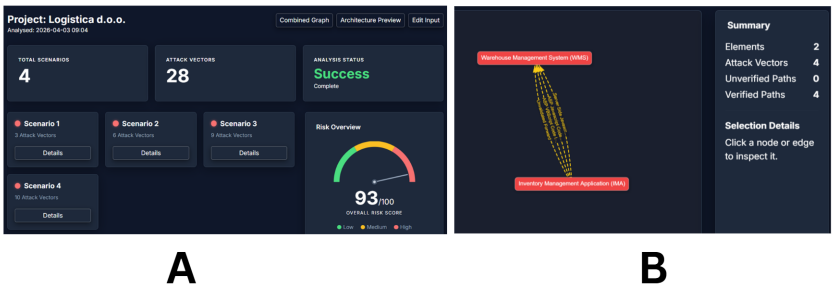


Figure 2: Image A: Prototype dashboard, Image B: Prototype attack vector roadmap.

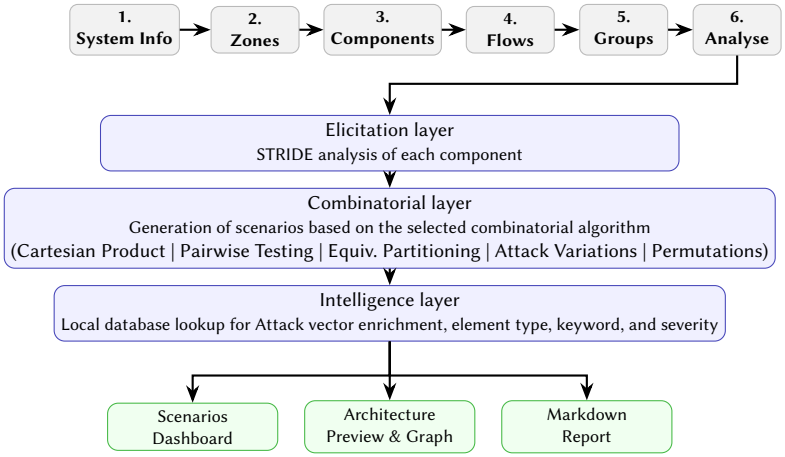


Figure 3: STRIDEX Analyser pipeline

Results are presented through three linked views - the scenario dashboard, architecture preview, and attack surface graph - alongside a downloadable Markdown report. Figure 2 shows two of these views: the scenario dashboard consolidates generated scenarios for inspection and review, while the attack vector roadmap visualises a selected scenario as a structured attack path, clarifying the sequence and relationship between threats.

STRIDEX is built around a small set of core entities that flow through a deterministic analysis pipeline. A System Model describes the target environment, including components, data flows, and trust boundaries, and can be created manually or via the Infrastructure Builder. Based on this, a Threat Profile is constructed as the complete set of identified threats, organised by component and mapped to the six fixed STRIDE categories. Each Threat represents a specific security issue with a description and countermeasures, forming the input to the analysis process. The system then generates Attack Scenarios, which are combinations of threats across categories to model multi-dimensional attacks. These scenarios are enriched using a database of Attack Vectors, which are real-world techniques mapped to STRIDE categories.

The process involves two human roles and two system components. The System Architect defines the system model and provides contextual metadata, while the Security Analyst creates or validates the threat profile, selects the analysis method, and evaluates the results. The STRIDEX analyser extends these two roles with two additional roles (two system components):

- Scenario generator: Takes the architecture and a strategy selection as input and produces a set of enriched attack scenarios.
- Attack intelligence: A collection of curated attack technique records in a database. Queried by the Scenario Generator during the enrichment phase.

The Scenario generator generates and enriches attack scenarios deterministically, and attack intelli-

gence serves as the source of attack-intelligence records. This separation supports reproducibility while reserving risk assessment and prioritisation for human judgement.

The workflow proceeds in five stages:

1. system modelling, where the architecture is defined,
2. threat elicitation, where the threat profile is created manually or automatically,
3. scenario generation, where the selected algorithm produces multi-category attack scenarios,
4. enrichment, where scenarios are annotated with relevant attack vectors and severity ratings and
5. analysis and mitigation, where the analyst reviews results and defines remediation actions

Figure 4 shows this workflow as a sequence diagram between the four roles in the STRIDEX analyser.

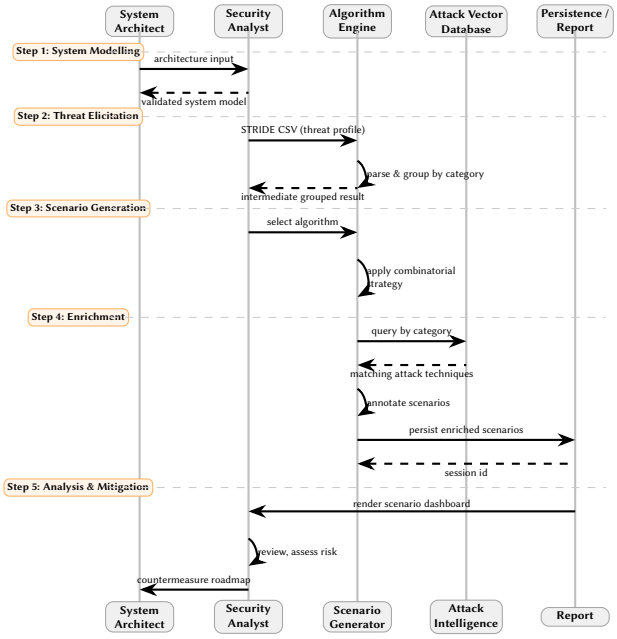


Figure 4: STRIDEX workflow

3. The Combinatorial Algorithms

All algorithms in this section are also implemented in the STRIDEX analyser and share the same processing pipeline: parse input, normalize threats by STRIDE category, generate scenarios with the selected combinatorial strategy, and enrich each scenario with matching attack-vector records. Table 3 summarizes the core equations used in this section and the interpretation of each expression.

To keep the main text concise, this section summarizes algorithm behavior, guarantees, and growth characteristics, while detailed step-by-step construction examples are provided in Appendix 4.

All five algorithms are discussed under the same normalized profile assumptions used throughout the comparison: active categories $\mathcal{K}$, category-specific threat counts n_i, and (for ordered-chain strategies) chain length ℓ with cap L.

3.1. Algorithm 1: Cartesian Product

The Cartesian Product algorithm generates every possible combination of one threat per active STRIDE category (Table 3). It provides exhaustive cross-category coverage under the selected category set, but output size grows multiplicatively as $\prod_{i \in \mathcal{K}} n_i$. For full worked construction and instantiated outputs, see Appendix A.2.

Table 3
Equation summary for the five combinatorial algorithms

Algorithm	Core equation	Interpretation
Shared symbols	$N_{(\cdot)}$	$N_{(\cdot)}$: number of generated scenarios
	$\mathcal{K}$	$\mathcal{K}$: active categories
	$m = \lvert\mathcal{K}\rvert$	m: number of active categories
	n_i	n_i: threats in category i
	ℓ	ℓ: chain length
	L	L: maximum chain length
Cartesian Product	$N_{\mathrm{CP}} = \prod_{i \in \mathcal{K}} n_i$	N_{CP}: total scenarios. Product multiplies per-category threat counts. Uniform case n^m means n choices repeated across m categories.
Pairwise Testing	$N_{\mathrm{PW}} \geq \max_{i<j}(n_i n_j)$	N_{PW}: number of pairwise scenarios. $n_i n_j$: value-pairs for one category pair (i, j). $\max$ selects the hardest pair; therefore this is a lower bound.
Equivalence Partitioning	$N_{\mathrm{EP}} = \lvert\mathcal{C}_{\mathrm{eq}}\rvert$	N_{EP}: retained representative scenarios. $\mathcal{C}_{\mathrm{eq}}$: equivalence classes. $\lvert\cdot\rvert$ is class count, so one representative per class.
Attack Variations	$N_{\mathrm{AVn^r}} = \frac{n!}{(n-r)!}$	N_{AV}: generated ordered chains and number of feasible attack variants is at most the sum of permutations.
Permutations	$N_{\mathrm{Pn}} = n!$	N_{P}: unconstrained ordered templates.

3.2. Algorithm 2: Pairwise Testing

Pairwise Testing guarantees that every pair of active STRIDE categories appears together in at least one scenario (Table 3). Compared with Cartesian Product, it reduces scenario volume while preserving 2-way interaction coverage, making it suitable when review capacity is constrained. For the worked covering-set construction, see Appendix A.3.orked construction and instantiated outputs, see Appendix A.2.

3.3. Algorithm 3: Equivalence Partitioning

Equivalence Partitioning produces compact output by grouping similar combinations into equivalence classes (Table 3). Its objective is representative diversity rather than exhaustive interaction coverage, so multiple concrete combinations can be intentionally collapsed into one retained scenario. For class definitions and representative outputs, see Appendix A.4.

3.4. Algorithm 4: Attack Variations

Attack Variations produces ordered multi-step attack sequences rather than unordered combinations.

Attack Variations produces ordered multi-step attack sequences rather than unordered combinations. It explicitly models temporal progression of exploitation steps, and therefore supports more realistic incident-style chains; the final scenario count depends on the variation heuristics used by the generator [10]. For chain construction examples, see Appendix A.5.

3.5. Algorithm 5: Permutations

The Permutations algorithm is the most exhaustive ordered-sequence strategy. Where Attack Variations selects a heuristic subset, Permutations enumerates all distinct ordered category templates up to a chosen length cap L, before practical filtering constraints are applied. For instantiated ordered-template outputs, see Appendix A.6.

3.6. Comparative Summary of Algorithms

This comparative subsection uses the same normalized running example (detailed in Appendix 4), so that output differences are directly attributable to algorithm behaviour rather than input variation.

To complement this comparison, we also exercised the same implementation on broader normalized STRIDE profiles derived from repository examples used by the tool workflow, and examples created to stress category and threat cardinalities. These profiles were not taken from an enterprise deployment; rather, they were constructed from two sources. First, we reused representative architecture examples from the STRIDEX analyser repository and the accompanying sample workflow artifacts, which capture common web-service and API-style components. Second, we created additional synthetic profiles by the authors to stress the number of components, active STRIDE categories, and threats per category. Those profiles ranged from 4 to 24 components, with 2 to 6 active STRIDE categories per component and 2 to 12 threats per active category. Across these runs, each strategy was executed under controlled conditions to measure scenario volume, category representation, and growth behaviour. As an illustrative baseline outside the tool itself, we also compared the same administrator-panel example using a conventional manual STRIDE review and the STRIDEX workflow. The manual review produced isolated per-category findings, whereas the STRIDEX workflow generated a compact scenario set that explicitly linked spoofing, tampering, information disclosure, and privilege escalation.

To keep the article within the conference page limit, this work remains a prototype-level evaluation; formal user studies with professional analysts and a head-to-head comparison against STRIDE are left for future work. The comparison between STRIDE and STRIDEX is illustrative rather than a full user study, but it highlights how the unit of analysis shifts from isolated threats to cross-category attack paths.

The weight vector was selected for this study as a governance-oriented profile rather than as a universal optimum; different organizations may adopt different weight sets. Interaction Coverage was assigned the highest weight, 25%, because the primary objective of STRIDEX is to preserve cross-category interaction coverage in scenario generation. Scalability, Reviewability, and Attack Realism were each assigned 20% because they represent closely related practical constraints on usable output size, analyst workload, and sequence fidelity. Compliance Fit was assigned 15% because it is an important but secondary decision factor in the comparison. Scores are assigned on a 1–5 scale (1 = poor, 5 = excellent). The purpose of this example is not to claim statistical significance or definitive superiority, but to provide a structured, reproducible decision aid for practitioners who need to choose an algorithm under a given governance objective.

In this revision, the grading was performed by the authors as a transparent demonstration procedure; objective scoring and a more detailed analysis are future work.

- Interaction Coverage (25%): Proportion of meaningful cross-category attack interactions guaranteed to appear in the output. (1: ad-hoc, 3: partial systematic, 5: formal exhaustive guarantee for target interaction level)
- Scalability (20%): How output volume grows as the number of components and active STRIDE categories increases. (1: infeasible for one-session review on large profiles, 3: manageable with substantial filtering, 5: consistently single-session)
- Reviewability (20%): Whether a practitioner can review the entire output in a single session without cognitive overload. (1: infeasible for one-session review on large profiles, 3: manageable with substantial filtering, 5: consistently single-session)
- Attack Realism (20%): Degree to which the algorithm models temporally ordered, multi-step exploitation sequences as they occur in real incidents. (1: no ordering, 3: partial sequence semantics, 5: explicit multi-step ordering)
- Compliance Fit (15%): Suitability for regulatory submissions audit trails, and evidence-of-due-diligence requirements. (1: weak audit traceability, 3: moderate, 5: strong evidence trail)

The weighted score is given by:

$$\bar{s} \;=\; 0.25\, c_{\text{cov}} + 0.20\, c_{\text{scl}} + 0.20\, c_{\text{rev}} + 0.20\, c_{\text{rel}} + 0.15\, c_{\text{cmp}} \tag{1}$$

To supplement the comparative analysis with concrete evidence, Appendix 4 uses the normalized running example with four active categories solely for exposition; the implementation supports profiles with a variable number of active categories, and the table is not meant to exhaust all STRIDE categories in a single sample. All examples were generated from the same normalized profile with four active categories (S, T, I, E) and two threats per active category, so that differences reflect algorithmic behaviour rather than input variation. he examples are abbreviated and intended to illustrate characteristic output structure.

Table 4 shows the per-criterion scores and the weighted aggregate values for all five algorithms. Cartesian Product achieves maximum coverage (5/5) through exhaustive enumeration but scores low on scalability and reviewability (2/5) due to rapid growth; realism is moderate (3/5) and compliance strong (5/5). Pairwise Testing shows coverage (4/5), scales efficiently (4/5), remains reviewable (4/5), but lacks ordering (3/5), with solid compliance (4/5). Equivalence Partitioning is highly scalable and reviewable (5/5) but has limited coverage (2/5), low realism (2/5), and weak compliance (2/5). Attack Variations balances diversity (3/5) with good scalability and reviewability (4/5), achieves high realism (5/5), but lacks formal coverage guarantees (3/5 compliance). Permutations provide strong ordered coverage (4/5) and realism (5/5), with moderate scalability and reviewability (3/5), and good compliance support (4/5).

Differences in $\bar{s}$ should be interpreted as ordinal decision support, not as statistically significant performance gaps. In particular, small differences should be treated as practical ties and resolved by context-specific priorities.

Table 4
Multi-criteria scores for the five algorithms.

Algorithm	Interaction Coverage	Scalability	Reviewability	Attack Realism	Compliance Fit	$\bar{s}$
Cartesian Product	5	2	2	3	5	3.40
Pairwise Testing	4	4	4	3	4	3.80
Equiv. Partitioning	2	5	5	2	2	3.20
Attack Variations	3	4	4	5	3	3.80
Permutations	4	3	3	5	4	3.80

As coverage increases, output volume generally increases as well, which directly affects analyst workload, review time, and the consistency of downstream prioritization. A strategy can therefore be selected, but only relative to explicit governance priorities and project constraints rather than as a universal optimum. The appropriate choice depends on the review capacity of the team, compliance requirements and whether the temporal order of exploitation steps carries analytical significance [11, 12]. In practical terms, algorithm selection should be treated as a governance decision rather than only a technical one, because the selected strategy determines not only what interactions are discovered but also what can be reviewed credibly within project constraints.

Context-specific overrides remain appropriate. Cartesian Product is justified for small-scope, strict-compliance audits where exhaustive evidence is required despite higher review burden. Equivalence Partitioning is appropriate for early-design rapid triage, where broad semantic coverage is preferred over exhaustive interaction enumeration. Attack Variations and Permutations are preferable when a red-team or incident-response mandate requires explicit, documented exploitation sequences, including multi-step chains such as four-step advanced persistent threat progressions [13].

4. Discussion and Conclusion

This paper presented STRIDEX, an extension of the STRIDE threat modeling methodology that replaces flat, per-category threat lists with structured multi-category attack scenarios produced by combinatorial

algorithms from software testing theory. The paper maps combinatorial testing concepts to STRIDE analysis: STRIDE categories become parameters, concrete threats become parameter values, and the chosen algorithm determines which interactions are represented in the output. Five algorithms were presented and analysed: Cartesian Product, Pairwise Testing, Equivalence Partitioning, Attack Variations, and Permutations. In these prototype evaluation, Pairwise Testing, Attack Variations and Permutations provided balanced trade-offs across the selected criteria. The contribution of STRIDEX is a decision-support that complements STRIDE rather than replacing it. It helps analysts explore cross-category attack paths in a repeatable way, while still requiring human judgement for prioritisation and mitigation.

This work also has several scope limitations. Although STRIDEX, as a model and prototyping tool, covers a broader end-to-end process, this article focuses only on the combinatorial design of complex multi-category scenarios and their validation. Other dimensions, such as comparative benchmarking against STRIDE and other existing threat-modeling tools, controlled analyst studies, long-term operational integration, and measured downstream risk-reduction impact, are not yet evaluated in detail here. Therefore, the findings should be understood as preliminary evidence regarding scenario creation behaviour, not as a comprehensive validation of the entire system. At the same time, the results must be interpreted with caution: because combinatorial generation can establish links between threats that are formally compatible but operationally implausible, some of the generated scenarios may be nonsensical or analytically irrelevant. Such scenarios should therefore be removed through filtering, constraints, or analyst review before the output is used as a basis for risk assessment.

Future work will focus on empirical validation of the multi-criteria scoring model against real-world engagement data, direct comparison with STRIDE and related tools in contolled case studies, and integration with continuous delivery pipelines to support automated analysis on architecture change events.

Acknowledgments

The authors acknowledge the financial support from the Slovenian Research Agency (Research Core Funding No. P2-0057).

Declaration on Generative AI

Authors declare that AI tools were used solely for translation purposes and to improve language clarity. All content, ideas, and conclusions are author's own, and the translated material has been reviewed for accuracy.

References

[1] T. M. Connect, State of threat modeling 2024-25, 2025. URL: https://www.threatmodelingconnect.com/state-of-threat-modeling-2024-25.

[2] N. Shevchenko, T. A. Chick, P. O'riordan, T. P. Scanlon, C. Woody, Threat modeling: a summary of available methods, Technical Report, 2018.

[3] C. MacKenzie, P. Akbari, M. Gabriel, G. N. Nur, K. Wall, An Analysis of Emerging Threats in Iowa: Laying the Foundations for Quantitative Risk Assessments, Technical Report, Iowa State University, 2024.

[4] N. Azam, A. L. Michala, S. Ansari, N. B. Truong, Modelling technique for GDPR - compliance: Toward a comprehensive solution, in: GLOBECOM 2023-2023 IEEE global communications conference, IEEE, 2023, pp. 3300–3305. doi:10.48550/arXiv.2404.13979.

[5] R. E. Thompson, M. McLaughlin, C. Powers, D. Votipka, "There are rabbit holes I want to go down that I'm not allowed to go down": An investigation of security expert threat modeling practices for medical devices, in: 33rd USENIX Security Symposium (USENIX Security 24), USENIX Association,

Philadelphia, PA, 2024, pp. 4909–4926. URL: https://www.usenix.org/conference/usenixsecurity24/presentation/thompson.

[6] P. D. Chowdhury, M. Sameen, J. Blessing, N. Boucher, J. Gardiner, T. Burrows, R. Anderson, A. Rashid, Threat models over space and time: a case study of E2EE messaging applications, arXiv preprint arXiv:2301.05653 (2023).

[7] J. Zhang, Z. Zhang, F. Ma, Introduction to combinatorial testing, SpringerBriefs in Computer Science 0 (2014) 1–16. URL: https://link.springer.com/chapter/10.1007/978-3-662-43429-1_1. doi:10.1007/978-3-662-43429-1_1/SAVE-RESEARCH.

[8] J. D. Hagar, T. L. Wissink, D. R. Kuhn, R. N. Kacker, Introducing combinatorial testing in a large organization, Computer 48 (2015) 64–72. doi:10.1109/MC.2015.114.

[9] L. Belloli, P. A. M. Mediano, R. Cofré, D. F. Slezak, R. Herzog, Thoi: An efficient and accessible library for computing higher-order interactions enhanced by batch-processing, PLOS ONE 21 (2026) 1–23. URL: https://doi.org/10.1371/journal.pone.0348005. doi:10.1371/journal.pone.0348005.

[10] J. Straub, Modeling attack, defense and threat trees and the cyber kill chain, att&ck and stride frameworks as blackboard architecture networks, in: 2020 IEEE International Conference on Smart Cloud (SmartCloud), 2020, pp. 148–153. doi:10.1109/SmartCloud49737.2020.00035.

[11] P.-A. Wang, K. Ariu, A. Proutiere, On universally optimal algorithms for a/b testing, arXiv preprint arXiv:2308.12000 (2023). doi:10.48550/arXiv.2308.12000.

[12] M. Bures, B. S. Ahmed, Employment of multiple algorithms for optimal path-based test selection strategy, Information and Software Technology 114 (2019) 21–36. URL: https://www.sciencedirect.com/science/article/abs/pii/S0950584919301399. doi:10.1016/J.INFSOF.2019.06.006.

[13] M. Mc Cabe, S. H. Houmb, Enhancing the mitre att&ck® framework for cyber-physical systems using insights from advanced persistent threats, Applied Sciences 16 (2026). URL: https://www.mdpi.com/2076-3417/16/4/1815. doi:10.3390/app16041815.

Appendix: Worked STRIDE-to-STRIDEX Example

This appendix provides a fully worked example using the same running profile as in Section 3: four active STRIDE categories $\{S, T, I, E\}$ with two concrete threats per category. The goal is to show, step by step, how each combinatorial algorithm constructs STRIDEX scenarios from the same threat profile. Some algorithms can generate more than four outputs for this profile; for simplification purposes, this appendix limits each algorithm illustration to four representative examples.

A.1 Example STRIDE threat profile (input)

The analysed component is an Internet-facing administrator panel with a backend API and logging subsystem. Table 5 lists the normalized STRIDE input used for all five algorithms.

A.2 Algorithm 1 (Cartesian Product): how output is created

constructed scenarios:

- CP-01: $S_1 + T_1 + I_1 + E_1$
- CP-02: $S_1 + T_1 + I_1 + E_2$
- CP-03: $S_1 + T_1 + I_2 + E_1$
- ....
- CP-16: $S_2 + T_2 + I_2 + E_2$

Created scenario text from these threats:

- CP-01: The attacker steals an administrator session cookie (S_1), modifies server configuration to weaken validation (T_1), reads backup tokens from exposed logs (I_1), and then uses the recovered token to obtain administrator privileges (E_1).
- CP-02: The attacker steals an administrator session cookie (S_1), tampers with server configuration (T_1), extracts backup tokens from logs (I_1), and escalates to super-admin rights via weak service-account policy and token replay (E_2).

Table 5
Worked STRIDE threat profile used as STRIDEX input

Cat.	ID	Concrete threat example
S	S_1	Attacker steals a valid administrator session cookie via phishing and reuses it to impersonate the admin user.
S	S_2	Attacker replays a captured OAuth bearer token from an unmanaged endpoint to authenticate as a privileged service account.
T	T_1	Attacker modifies a server-side configuration file to disable input validation and enable unsafe debug flags.
T	T_2	Attacker alters role-to-permission mappings in a database table, granting write access to unauthorized identities.
I	I_1	Attacker reads plaintext backup tokens from a misconfigured application log exposed through a diagnostics endpoint.
I	I_2	Attacker extracts user PII from an unencrypted backup archive stored in a publicly reachable object bucket.
E	E_1	Attacker uses a recovered backup token to obtain administrator privileges in the panel.
E	E_2	Attacker chains a weak service account policy with token replay to escalate from operator to super-admin rights.

- CP-03: The attacker steals an administrator session cookie (S_1), modifies server configuration (T_1), extracts PII from an unencrypted public backup archive (I_2), and then gains administrator privileges using recovered token material (E_1).
- ....
- CP-16: The attacker replays a captured OAuth bearer token (S_2), alters role-to-permission mappings in the database (T_2), extracts PII from a publicly reachable backup (I_2), and escalates from operator to super-admin rights (E_2).

A.3 Algorithm 2 (Pairwise Testing): how output is created

One valid minimal-size covering set (4 scenarios) is:

- PW-01: $S_1 + T_1 + I_1 + E_1$
- PW-02: $S_1 + T_2 + I_2 + E_2$
- PW-03: $S_2 + T_1 + I_2 + E_2$
- PW-04: $S_2 + T_2 + I_1 + E_1$
- ....

Created scenario text from these threats:

- PW-01: Cookie theft enables impersonation (S_1), configuration tampering weakens controls (T_1), exposed logs leak backup tokens (I_1), and token reuse yields administrator access (E_1).
- PW-02: Cookie theft (S_1) is followed by permission-table tampering (T_2), then extraction of PII from a public backup (I_2), and final escalation to super-admin rights (E_2).
- PW-03: OAuth token replay as a privileged service account (S_2) is combined with configuration tampering (T_1), public-backup disclosure (I_2), and escalation via weak service-account policy (E_2).
- PW-04: OAuth token replay (S_2), role-mapping tampering (T_2), leakage of backup tokens from logs (I_1), and privileged takeover through recovered token material (E_1).
- ....

A.4 Algorithm 3 (Equivalence Partitioning): how output is created

The classes are defined by dominant attacker intent:

- EQ-Class A (credential abuse + privilege gain)
- EQ-Class B (token replay + policy escalation)
- EQ-Class C (tampering-led data exposure)
- EQ-Class D (data exposure-led escalation)

Representative generated scenarios:

- EQ-01 (Class A): $S_1 + T_1 + I_1 + E_1$

- EQ-02 (Class B): $S_2 + T_2 + I_2 + E_2$
- EQ-03 (Class C): $S_1 + T_2 + I_1 + E_2$
- EQ-04 (Class D): $S_2 + T_1 + I_2 + E_1$

Created scenario text from these threats:

- EQ-01 (Class A): Stolen admin cookie (S_1) and server-configuration tampering (T_1) expose backup tokens in logs (I_1), enabling direct administrator takeover (E_1).
- EQ-02 (Class B): Replayed OAuth token (S_2) and permission-table manipulation (T_2) are followed by backup archive disclosure (I_2), ending in super-admin escalation (E_2).
- EQ-03 (Class C): Cookie-based impersonation (S_1) plus role-mapping tampering (T_2) amplifies log-based token leakage (I_1), resulting in high-privilege escalation (E_2).
- EQ-04 (Class D): Token replay (S_2) and configuration tampering (T_1) lead to PII disclosure from public backup storage (I_2), followed by administrator privilege gain (E_1).

A.5 Algorithm 4 (Attack Variations): how output is created

Ordered chains:

- AV-01: $S_1 \to T_1 \to I_1 \to E_1$
- AV-02: $S_2 \to I_1 \to E_2$
- AV-03: $T_2 \to I_2 \to E_2$
- AV-04: $S_1 \to T_2 \to E_1$
- ...

Created scenario text from these threats:

- AV-01: The attacker hijacks an admin session cookie (S_1), modifies server configuration to reduce defenses (T_1), extracts backup tokens from exposed logs (I_1), and elevates to administrator (E_1).
- AV-02: The attacker replays a captured OAuth token (S_2), harvests backup tokens from logs (I_1), and escalates to super-admin through weak service-account controls (E_2).
- AV-03: The attacker first alters role-permission mappings (T_2), then extracts PII from public backup storage (I_2), and finally escalates to super-admin rights (E_2).
- AV-04: The attacker steals an administrator session cookie (S_1), tampers with role mappings (T_2), and then converts that position into administrator privileges (E_1).
- ...

A.6 Algorithm 5 (Permutations): how output is created

Instantiated outputs:

- PM-01 template (S, T): $S_1 \to T_1$
- PM-02 template (T, I, E): $T_2 \to I_1 \to E_2$
- PM-03 template (E, S, I, T): $E_1 \to S_2 \to I_2 \to T_1$
- PM-04 template (I, T, S, E): $I_1 \to T_1 \to S_1 \to E_2$
- ...

Created scenario text from these threats:

- PM-01: The attacker impersonates an administrator through stolen session credentials (S_1) and then tampers with server configuration (T_1).
- PM-02: The attacker modifies role mappings (T_2), uses diagnostics leakage to obtain backup tokens (I_1), and escalates to super-admin rights (E_2).
- PM-03: Starting from a privileged foothold (E_1), the attacker replays service-account credentials (S_2), extracts PII from public backups (I_2), and alters configuration state for persistence (T_1).
- PM-04: The attacker first obtains sensitive tokens from logs (I_1), tampers with configuration to suppress safeguards (T_1), impersonates an administrator via cookie theft (S_1), and completes escalation to super-admin (E_2).
- ...