Counterfactual Defense Synthesis through 3-Way Timeline Merging for Branched Simulations

Damijan Novak*,†, Iztok Fister Jr. and Jani Dugonik†

University of Maribor, Faculty of Electrical Engineering and Computer Science, Koroška cesta 46, 2000 Maribor, Slovenia

Abstract

Analysts investigating multistage cyberattacks face a fundamental tension. To evaluate defensive strategies they must compare how the same simulated incident would unfold under different response choices, but simulation state cannot be reconciled across branches by ad-hoc file diffing (i.e., by manually comparing and merging checkpoint files or logs from divergent simulator snapshots). We present a 3-way merge framework for branched simulation timelines that adapts software engineering merge formalisms to runtime simulation state. The framework contributes: (1) a conflict taxonomy comprising seven types dimensioned for simulation semantics, (2) a declarative resolution policy engine that expresses analyst preferences as (conflict type, property key, resolution) triples, and (3) a case study on a scripted multistage attack evaluated across five scenario variants. Merging guided by declarative policies produces hybrid timelines that Pareto dominate both analyst response strategies on state metrics relevant to safety in all four experimental variants, under the precondition that analyst preferences are separable per property and the metrics monotone in the values of each property. Pure conservative merging fails to produce a usable timeline under the same construction, illustrating the structural asymmetry between the lowest common ancestor merge base and post branch state that motivates the declarative policy approach.

Keywords

cybersecurity simulation, 3-way merge, counterfactual analysis, timeline branching, conflict resolution, agent-based simulation

1. Introduction

Analyst evaluation of cyber incident response increasingly relies on what-if exploration using simulations. Even with research gyms for red/blue agents, operational defence often still depends on human analysts, exposing a gap to practice [1]. Digital-twin-based incident response [2] takes one step toward closing this gap by allowing analysts to evaluate alternative response strategies in a parallel environment. The remaining gap is that comparison between alternatives is performed by inspecting metrics on each fork independently, not by synthesising a hybrid trajectory that combines the safety-relevant gains of one branch with those of another. The analyst's actual decision is rarely binary. A fast strategy may catch most attackers but exhaust monitoring quotas. A thorough strategy may miss the first wave but conserve resources for later stages. The analyst wants the per-property combination of both, resulting in a synthesised timeline that no single branch produced.

This synthesis is a counterfactual reasoning problem (Pearl's third causal-hierarchy rung [3]), which we frame computationally as a 3-way merge. The pre-branch trajectory is a common ancestor, the two candidate strategies are sibling branches, and the desired output is a merged trajectory that preserves both sets of analyst intent at the granularity of individual entities and time steps. The version-control community has worked on 3-way merge since the 1980s [4], and recent work has extended the primitive to relational data, parametric computer-aided design (CAD) models, and Eclipse Modeling Framework artifacts. None of these formalisms target simulation runtime state. The closest prior work on simulation merge, Hybinette and Fujimoto (2001) [5], introduces simulation cloning at decision points to enable parallel what-if exploration but runs clones forward independently and does not address reconciliation

SQAMIA 2026: 13th Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications, September 16–18, 2026, Niš, Serbia

*Corresponding author.

†These authors contributed equally.

damijan.novak@um.si (D. Novak); iztok.fister1@um.si (I. F. Jr.); jani.dugonik@um.si (J. Dugonik)

© 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/paper11.pdf

of divergent clone state.

This article makes three contributions. First, a seven-type conflict taxonomy specific to simulation state, covering entity existence, state values, spatial position, temporal order, lifecycle events, structural shape, and semantic compatibility. Second, a declarative resolution policy engine that expresses analyst preferences as triples of the form (conflict type, property key, resolution), separating the domain-specific question of what to prefer when branches disagree on a property from the domain-independent merge mechanism. Third, a case study across five scenario variants in which merging that is driven by policy Pareto dominates both source branches on safety-relevant state metrics in all four experimental variants, while pure conservative merging fails to produce a usable timeline under the same input.

The rest of the article is organised as follows. Section 2 surveys related work on 3-way merge formalisms, counterfactual reasoning in cybersecurity, and simulation cloning, positioning our contribution at their intersection. Section 3 introduces the timeline and branching model. Section 4 defines the seven-type conflict taxonomy and the merge algorithm. Section 5 describes the implementation, including the resolution policy engine. Section 6 reports the case study results. Section 7 discusses what the results do and do not generalise to. Section 8 concludes with directions for follow-up work.

2. Related Work

2.1. 3-way merge formalisms

The 3-way merge primitive underlies version control and any tool that reconciles concurrent edits. Mens [4] catalogued syntactic, structural, and semantic software-merge conflicts in the context of source code and design models. The primitive has since been extended to relational tables (cell-level merge in Dolt [6]) and to design artifacts (Onshape [7], EMF Compare [8]). What unites these is a fundamentally *static* artifact (a source file, a relational row, a CAD feature, a model) whereas simulation runtime state is a function over time: an entity's property can differ between tick 50 and tick 51 on both branches. Conflict detection must therefore operate over a time-indexed product space rather than a flat key set, and resolution must respect the temporal continuity each branch records. None of [4, 6, 7, 8, 9, 10] addresses this directly.

2.2. Counterfactual reasoning in cybersecurity

Cybersecurity what-if platforms typically support comparison through ad-hoc replay rather than structured merge. CybORG and the CAGE Challenge series [11, 12] provide multi-agent RL environments in which defender policies are compared by running the simulation separately under each. Hammar et al. [13] apply Pearl-style causal counterfactual reasoning to CAGE-2 through tree search over single trajectories, and YAWNING TITAN [14] takes a similar training-environment approach. Gyevnár et al. [15] (AXIS) generate single-trajectory counterfactuals, and Allison et al. [2] evaluate response strategies on a cyber-physical digital twin. In all of these, alternatives are compared by inspecting metrics on each fork independently. *Synthesising* a hybrid timeline that combines both is out of scope [2, 15].

2.3. Simulation cloning and forking

Simulation cloning and forking are well studied in parallel discrete-event simulation. The closest prior work, Hybinette and Fujimoto [5], clones at decision points but runs clones forward independently without reconciling divergent state; later work extends forward-only cloning (e.g., GPU agent simulation [16]). Reconciling deliberately divergent trajectories (i.e., the topic of this article) is not addressed in that literature.

2.4. Positioning

Our contribution sits at the intersection of these three literatures: from structured merge [4, 6, 7, 9, 8, 10] we take the (base, branch_1, branch_2) merge structure with its detection and resolution steps; from

cybersecurity what-if [2, 11, 12, 13, 14, 15, 17] the analyst decision-point use case and the requirement to preserve safety-relevant semantics; and from simulation cloning [5, 16] the runtime-state representation and live forking. None provides what our framework does: a 3-way merge primitive over simulation runtime state with a domain-specific conflict taxonomy and a declarative resolution policy. The Pareto-dominance result (Section 6) draws on multi-objective RL for cyber defence [18] and classical counterfactual reasoning [3]. The analyst-loop deployment gap is surveyed by Vyas et al. [1].

3. Timeline and Branching Model

3.1. Timeline

We define a Timeline as a time-indexed record of simulation state and a BranchableTimeline as a Timeline embedded in a directed acyclic graph of parent-child relations. Together with the Lowest Common Ancestor (LCA) operation defined in 3.3, these constructions provide the (base, branch1, branch2) triple that the 3-way merge algorithm in Section 4 consumes. Let E be a finite set of entity identifiers and let P be the set of property maps, where each property map is a partial function from property names (strings) to typed values. A Keyframe at tick t is a partial function $k : E \to P$ assigning to each entity present at t the property map describing its state. A Timeline is a tuple $T = (T, E, K)$ where $T \subset \mathbb{N}$ is a finite, totally ordered set of ticks (the temporal extent of the timeline), E is the entity population, and $K : T \to (E \to P)$ is the keyframe function that maps each tick to a Keyframe. We write k_t for $K(t)$ and write $k_t(e).p$ for the value of property p on entity e at tick t, leaving $k_t(e).p$ undefined when entity e has no record at tick t.

An entity may have no record at some tick (because the entity does not yet exist, has been retired, or because the simulator did not capture state at that tick), so the merge algorithm must distinguish a missing entry from a present entry whose value happens to be null.

3.2. BranchableTimeline

A BranchableTimeline B is a pair (T, parent) where T is a Timeline as defined in 3.1 and parent is either undefined (B is a root) or a reference to another BranchableTimeline B_p whose initial keyframes (ticks $0..t_b$ for some branch tick t_b) coincide with the initial keyframes of B. The BranchableTimeline population of an experiment forms a directed acyclic graph rooted at one or more parentless timelines, with edges directed from child to parent (each child stores a pointer to its parent, mirroring the version-control parent-pointer convention). The branch operation, $\mathrm{branch}(B, t_b)$, takes an existing BranchableTimeline B and a branch tick t_b in B's temporal extent, and produces a fresh BranchableTimeline B' whose initial keyframes are copied from B at ticks $0..t_b$ and whose parent pointer references B. Subsequent keyframes added to B' record state produced by an alternative simulation continuation from the snapshot at t_b.

3.3. Branch points and the Lowest Common Ancestor

Given two sibling BranchableTimelines B_1 and B_2 with common parent B_p, the branch point of (B_1, B_2) is the tick t_b at which B_1 and B_2 were forked from B_p. The LCA of (B_1, B_2) is defined as the truncation of B_p to the prefix $[0..t_b]$. The LCA is the latest state both branches share. It serves as the base input to the 3-way merge algorithm in Section 4, with B_1 and B_2 supplying the two alternatives to reconcile.

Since the LCA covers only $[0..t_b]$, every conflict at a tick $t > t_b$ has an undefined base value, a structural asymmetry that 6.5 reports empirically and 7.2 generalises beyond the case study.

4. 3-Way Merge Algorithm and Conflict Taxonomy

4.1. Conflict taxonomy

This section defines the merge algorithm. The algorithm consumes the (B_p, B_1, B_2) triple constructed in Section 3 and produces a merged BranchableTimeline B_m by way of a three-phase pipeline. A conflict is a witness that the two siblings B_1 and B_2 disagree about some aspect of state at some post-branch tick. Table 1 defines the seven conflict types the detector recognises, where t denotes a tick with $t > t_b$ and k_t^i denotes $K^i(t)$ of branch B_i.

Table 1
Conflict types recognised by the detector. t denotes a tick with $t > t_b$ and k_t^i denotes $K^i(t)$ of branch B_i.

Type	Definition
ENTITY_EXISTENCE	exactly one of $k_t^1(e)$, $k_t^2(e)$ is defined for some t, e
STATE_VALUE	$k_t^1(e).p \neq k_t^2(e).p$ for some t, e, p where both are defined
SPATIAL_POSITION	STATE_VALUE specialised to position-typed properties
TEMPORAL_ORDER	the same pair of state transitions occurs in opposite order
LIFECYCLE	smallest or largest tick at which entity e appears differs
STRUCTURAL	temporal extents differ, or keyframe topology differs
SEMANTIC	user-supplied predicate $\varphi(B_1, B_2)$ flags incompatibility

The first six categories are mechanically detectable from the timeline data alone. The seventh is intentionally open and requires the integrating application to provide the predicate. The case study reported in Section 6 exercises only STATE_VALUE because the Fast-vs-Thorough defender choice modifies per-entity property values without altering entity population, lifecycle, position, event ordering, or timeline shape. The seven types are derived systematically rather than enumerated ad hoc. Starting from the classical software-merge categories of Mens [4] (structural and semantic conflicts) we specialise them to the axes along which two post-branch trajectories can disagree once state is a time-indexed function over (entity, property, tick) rather than a flat artifact. Each mechanically-detectable type isolates one axis: entity presence (ENTITY_EXISTENCE), property value (STATE_VALUE) and its spatial specialisation (SPATIAL_POSITION), the order of state transitions (TEMPORAL_ORDER), entity lifespan (LIFECYCLE), and timeline shape (STRUCTURAL). The seventh (SEMANTIC) retains Mens's semantically-detectable category as an application-supplied predicate. The upgrade over prior taxonomies is exactly this temporal specialisation: cell-level and model-merge taxonomies [6, 4, 10] target static artifacts over a flat key set and therefore have no analogue of the TEMPORAL_ORDER and LIFECYCLE axes, which arise only because simulation state is indexed over time. Section 7.3 outlines scenarios that would exercise the other categories.

4.2. Merge pipeline

The merge algorithm runs in three phases. Phase 1 (detection) walks the post-branch ticks of both siblings and collects every conflict witness into a set C. Phase 2 (resolution) applies the policy engine described in 4.3 to each conflict in C, producing a deterministic resolution decision for each. Phase 3 (merge) constructs the merged BranchableTimeline B_m by copying pre-branch keyframes from B_p, passing through agreed properties, and applying resolved values to conflicted ones. Phase 3 raises an exception if any conflict in C remains unresolved after Phase 2, which the rule grammar in 4.3 prevents by requiring a fall-through rule.

4.3. Resolution strategy grammar

A resolution rule is a triple (type, key, verb) where type is a conflict type, key is a property name or the wildcard $*$, and verb is one of eight resolution verbs. The policy engine evaluates rules in order. The first rule whose type and key fields match the conflict's type and propertyKey determines the verb to

apply. A wildcard fall-through rule with type $= *$ and key $= *$ guarantees that every detected conflict receives a resolution before Phase 3 runs. Each verb is a function $f : (v_p, v_1, v_2) \to v$ that takes the base value v_p (the LCA value, possibly undefined as discussed in 3.3) and the two branch values v_1 and v_2. Table 2 lists the eight verbs.

Table 2
Resolution verbs implemented natively by the policy engine.

Verb	Returns
TAKE_BRANCH1	v_1
TAKE_BRANCH2	v_2
KEEP_BASE	v_p
TAKE_PRESENT	v_1 if defined, otherwise v_2, with v_p as fallback
PREFER_TRUE	true if $v_1 =$ true or $v_2 =$ true, else false
PREFER_FALSE	dual of PREFER_TRUE
NUMERIC_MIN	$\min(v_1, v_2)$
NUMERIC_MAX	$\max(v_1, v_2)$

The expressive power comes from composing rules, not from rich verbs: the verbs are small total functions on the three inputs (Table 2), so any (rule, conflict) pair produces a defined output, and the guaranteed wildcard fall-through makes Phase 3's no-unresolved-conflict precondition a structural property rather than a runtime check. The rule file (5.2) is thus the entire, externally inspectable policy specification for a given experiment, and the verb set is extensible by registering new functions.

5. Implementation

5.1. Architecture

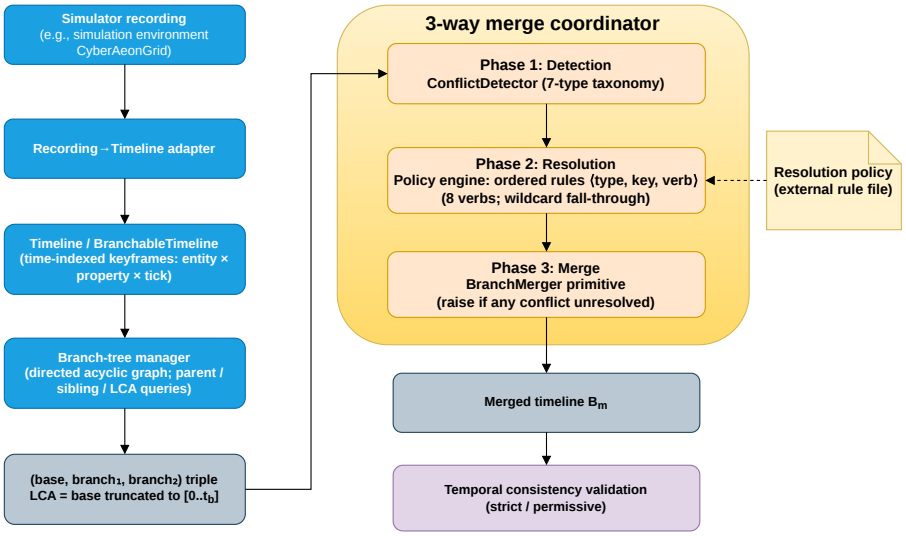


Figure 1: Framework architecture: a simulator recording is adapted into a branchable timeline, the branch-tree manager supplies the (B_p, B_1, B_2) triple to the three-phase merge coordinator (detection, policy resolution, merge), and the merged timeline B_m is checked by temporal-consistency validation. The external rule file drives the resolution policy engine (Phase 2).

The framework is organised into cooperating modules (Figure 1) that separate the timeline data model and its branchable extension, conflict detection (the seven-type detector of 4.1, each conflict paired with its resolution), the resolution policy engine, and a merge coordinator that drives detection, resolution, and merge in sequence.

Integration with existing simulators is mediated by a recording-to-timeline adapter and a branch-tree manager that owns the directed acyclic graph of timelines and exposes parent, sibling, and LCA queries. Temporal consistency validation operates in two modes: a strict mode that requires every state transition to be paired with an explicit event, and a permissive mode that flags only critical inconsistencies. The case study uses the permissive mode because the underlying simulator does not emit explicit recovery-action events.

5.2. Resolution policy engine

The resolution policy engine is configured at runtime from an external rule file shipped with the deployed distribution. Each rule is identified by a numeric index and specified by three fields: a conflict type, a property key, and a resolution. The engine groups fields by index, loads the rules into an ordered list, and matches each detected conflict against the list in order, applying the first rule whose conflict type and property key match. Wildcard matching is supported with the literal token "$*$" in either field.

Verbs referenced in a rule but not implemented fall through to a base-value default, ensuring policy files cannot produce undefined behaviour.

The rule list applied in the case study (Table 3) consists of nine entries: five address STATE_VALUE flags modified by the defender agents (`compromised`, `defended`, `isolated`, `monitored`, `vulnerabilityScore`), three cover ENTITY_EXISTENCE, STRUCTURAL, and LIFE-CYCLE conservatively, and a wildcard catch-all at row 9 guarantees termination of the resolution pass.

6. Case Study: Counterfactual Defense Synthesis

This section evaluates our 3-way merge framework on a scripted multi-stage attack scenario, varied across five parameterizations. The central experimental question is whether policy-driven merging of two analyst defense choices can produce a hybrid timeline that Pareto-dominates either individual choice on safety-relevant metrics. A secondary question is whether the naive conservative merge, commonly used as a baseline in version-control systems, suffices on the same input, or whether declarative resolution policy is strictly required.

6.1. Scenario and protocol

The scenarios are instantiated in CyberAeonGrid, an internal research environment for cyber-defence simulation that hosts multiple families of decision-making agents (both our own implementations and open-source baselines) over a configurable network model. Here it supplies the scripted attacker and the standard/fast/thorough defenders and records their per-tick state as the timelines reconciled below. Each variant instantiates a random network topology of N nodes and runs a scripted attacker agent for $T = 150$ ticks. A branch point is fixed at tick $t_b = 65$: before t_b a standard defender agent is active. At t_b the experiment forks into two sibling trajectories representing two analyst response strategies. The fast branch introduces an aggressive defender (scan radius 2 hops, high clean probability) and the thorough branch introduces a conservative defender (scan radius 1 hop, 3-tick confirmation before cleanup). Both branches run to T from identical snapshot state at t_b. The pre-branch segment $(0..t_b)$ is the LCA and serves as the base input to the 3-way merge. The branch segments $(t_b..T)$ are the two alternatives to reconcile.

Five scenario variants exercise different network sizes and attack types. V1 (20 nodes, portscan), V2 (40 nodes, ransomware), V3 (60 nodes, DDoS), and V5 (40 nodes, lateral movement) are experimental variants with real analyst choice. V4 (40 nodes, data exfiltration) is a control variant in which the post-branch trajectory is identical on both sides, so the expected result is zero conflicts and an idempotent merge.

The framework is implemented in Java 22 as a Maven project.

6.2. Merge configurations

For each variant we produce five timelines: the base ($0..t_b$), the two branches (fast and thorough, each $0..T$), and two merged timelines covering the full interval $0..T$. The auto_merged timeline is produced by the framework's low-level merger under a conservative strategy that resolves every conflict to the base value. The policy_merged timeline is produced by the declarative policy engine (5.2) applying the rule set in Table 3.

Table 3
Resolution policy applied by the policy engine to compute policy_merged.

#	ConflictType	PropertyKey	Resolution	Rationale
1	STATE_VALUE	compromised	PREFER_FALSE	trust any branch reporting node clean
2	STATE_VALUE	defended	PREFER_TRUE	accumulate defensive credit
3	STATE_VALUE	isolated	PREFER_TRUE	prefer containment
4	STATE_VALUE	monitored	PREFER_TRUE	accumulate monitoring coverage
5	STATE_VALUE	vulnerabilityScore	NUMERIC_MIN	adopt lower-risk estimate
6	ENTITY_EXISTENCE	*	TAKE_PRESENT	include entities added by either branch
7	STRUCTURAL	*	KEEP_BASE	conservative on timeline shape
8	LIFECYCLE	*	KEEP_BASE	conservative on entity lifespan
9	*	*	KEEP_BASE	fall-through (ensures all resolved)

6.3. Conflict detection

Across the four experimental variants the detector reports 382 (V1), 2273 (V2), 2512 (V3), and 1977 (V5) conflicts. All conflicts are of type STATE_VALUE. The other six taxonomy categories do not trigger on Fast-vs-Thorough analyst branches because the two defender agents modify per-node flags on pre-existing entities without altering entity populations, spatial position, entity lifespan, or timeline topology.

Section 7.3 outlines scenarios that would exercise the remaining categories. Within STATE_VALUE, conflicts split exclusively between two property keys, `compromised` and `defended` (Figure 2).

The V4 control produces 0 conflicts, confirming that the detector does not hallucinate conflicts on identical branches and that the merge is idempotent in the degenerate case. The V4 auto-merged timeline is bit-identical to both source branches on all keyframes.

6.4. Pareto analysis

We evaluate five state-based metrics on each of the five timelines: peak compromised ratio (M1), final compromised ratio (M2), final defended ratio (M3), mean compromised ratio (M4), and time-to-first-recovery in ticks (M5). For M1, M2, M4, and M5 lower values are better. For M3 higher values are better. One timeline Pareto-dominates another if it is at least as good on all five metrics and strictly better on at least one, with a 5% noise floor for tied comparisons. Table 4 reports the verdicts. The policy-merged timeline Pareto-dominates the fast branch and the thorough branch in all four experimental variants. The margin is largest in V3 (DDoS), where policy_merged dominates thorough on all five primary metrics. Table 5 reports the underlying metric values. Figure 3 visualises the recovery gap directly. The V4 control shows NON_DOMINATED on all three comparisons with every metric tied, as expected from identical branches.

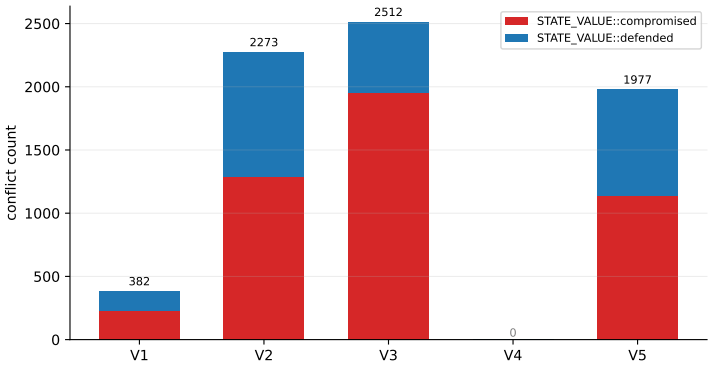


Figure 2: Conflict distribution by property key across variants. Stacked bar chart showing per-variant counts of STATE_VALUE::`compromised` (red) vs. STATE_VALUE::`defended` (blue) conflicts. V4 (control) shows zero conflicts as expected from identical branches.

Across Table 5, policy_merged reports the lowest or tied-lowest M2 in every variant and an M3 that matches the fast branch exactly. This reflects Rule R1 (STATE_VALUE::compromised → PREFER_FALSE) driving M2 down and Rule R2 (STATE_VALUE::defended →PREFER_TRUE) keeping M3 high, independently per property. This is the central property the article claims for declarative 3-way merge.

Table 4
Per-variant Pareto verdicts. "A" = policy_merged. INCOMPARABLE denotes that auto-merge failed to produce a timeline, so no meaningful per-metric comparison is possible.

Variant	vs fast	vs thorough	vs auto	Note
V1	A_DOMINATES	A_DOMINATES	INCOMPARABLE	portscan, 20 nodes
V2	A_DOMINATES	A_DOMINATES	INCOMPARABLE	ransomware, 40 nodes
V3	A_DOMINATES	A_DOMINATES	INCOMPARABLE	DDoS, 60 nodes
V4	NON_DOMINATED	NON_DOMINATED	NON_DOMINATED	control (identical branches)
V5	A_DOMINATES	A_DOMINATES	INCOMPARABLE	lateral movement, 40 nodes

Table 5
Primary metrics for the post-branch timelines per variant. M1 = peak compromised ratio, M2 = final compromised ratio, M3 = final defended ratio, M4 = mean compromised ratio, M5 = time-to-first-recovery (ticks).

Var	Timeline	M1	M2	M3	M4	M5
V1	fast	0.150	0.050	0.850	0.066	4
V1	thorough	0.150	0.100	0.750	0.091	4
V1	policy_merged	0.150	0.000	0.850	0.040	4
V2	fast	0.800	0.075	0.950	0.395	21
V2	thorough	0.800	0.525	0.525	0.549	21
V2	policy_merged	0.800	0.050	0.950	0.364	21
V3	fast	0.333	0.017	0.750	0.163	11
V3	thorough	0.383	0.383	0.583	0.331	150
V3	policy_merged	0.333	0.000	0.750	0.138	11
V4	fast = thorough = auto = policy	0.350	0.325	0.475	0.201	98
V5	fast	0.550	0.075	0.975	0.248	41
V5	thorough	0.550	0.400	0.650	0.397	41
V5	policy_merged	0.550	0.025	0.975	0.228	41

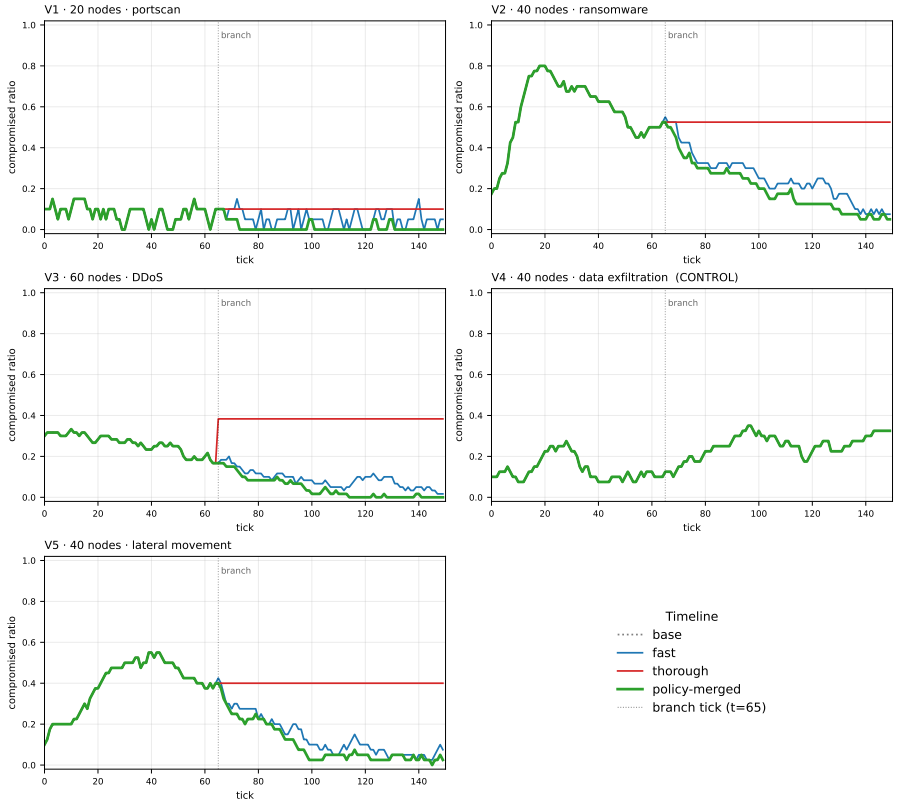


Figure 3: Compromised node ratio over time, per scenario variant. Fast (blue), thorough (red), and policy-merged (green, bold). All three timelines share the pre-branch prefix from the common base. They diverge only at the branch tick $t_b = 65$, marked with a vertical dotted line.

6.5. Auto-merge failure as an empirical finding

The auto_merged timeline fails to be produced for all four experimental variants, a predictable consequence of the conservative strategy on this input. Every STATE_VALUE conflict resolves to the base value, which is undefined past t_b, so the merge cannot construct the post-branch state and raises an exception. We report this as an empirical illustration of the LCA-asymmetry derived in 3.3 rather than a defect of conservative merge in general. The merged value must instead come from the branches under analyst-controlled rules, which the framework's declarative policy provides, producing hybrids that Pareto-dominate either source branch on the chosen metrics (Table 4).

All five timelines per variant pass temporal consistency validation in the permissive mode. The strict mode reports issues from a single rule that requires an explicit recovery-action event whenever a node transitions from compromised to clean, which the current simulator does not emit, so the failure affects all timelines equally and is reported as a simulator-instrumentation artifact rather than a merge-quality signal.

7. Discussion

The case study supports the central claims of the article, but several aspects of the result deserve elaboration, both to sharpen the contribution and to delimit what it does not claim. We organise the discussion around four threads: the interpretation of the Pareto result, the auto-merge failure as a finding about naive 3-way merge, the exercised scope of the conflict taxonomy, and threats to validity.

7.1. What the four out of four Pareto result means and does not mean

The observed dominance of policy_merged over both source branches in all four experimental variants is stronger than what the pre-registered protocol required, and the mechanism behind it is transparent: the per-property resolution rules (Table 3) independently optimise M2 and M3 by selecting the safer branch value per conflict, without a global compromise step. Because M2 and M3 are monotone in the property values the rules select, the per-property dominance is structurally guaranteed under these rules. The empirical contribution is that the structural guarantee transfers to scenario-level Pareto dominance across all four experimental variants without producing internally inconsistent timelines that fail temporal validation. We treat four out of four dominance as evidence that the framework operates correctly under the preconditions of separable per-property preferences and monotone metrics, not as a claim of broad superiority over alternative merge strategies. Comparison with whole-branch and prefer-present baselines under cost-aware metrics is left for future work.

The result does not claim that every pair of analyst choices, on every scenario, will admit a Pareto-dominant policy-merged hybrid. When the two branches encode non-separable trade-offs, for example a single action that both compromises a node and unlocks a recovery path downstream, per-property resolution cannot dominate either branch because the global outcome is path-dependent in a way that rule-based merging cannot capture. Our scenarios do not exhibit such couplings by construction: defender cleanup of a node is local in both branches and does not create downstream dependencies between property keys. Scenarios with coupled state, such as adversarial game trees or multi-step attack kill chains in which defense of node A causes the attacker to pivot to node B in only one branch, would require a richer resolution formalism, possibly one that incorporates lookahead. We treat this as a separate research question.

7.2. Auto-merge failure and LCA asymmetry

Conservative "keep base" resolution fails in forked simulations because the base is the LCA and contains no post-branch values. When branches disagree after t_b, "keep base" yields undefined values, so a merge cannot construct a complete post-branch state. This is not an implementation artifact: it follows whenever the merge base is LCA-local and post-branch observations are novel on both branches. Declarative per-property rules address the asymmetry by specifying how to combine branch values without depending on the base, producing semantically meaningful hybrid timelines. We leave systematic comparison against single-preference baselines to future work.

7.3. Taxonomy scope

The case study triggers only STATE_VALUE. Future work will evaluate scenarios that exercise ENTITY_EXISTENCE, LIFECYCLE, SPATIAL_POSITION, TEMPORAL_ORDER, and STRUCTURAL conflicts (e.g., sensor insertion, lifecycle shifts, topology rewiring, and event reordering).

7.4. Threats to validity

Four threats are material to generalisability. The evaluation is bound to a single simulator (CyberAeonGrid), with cross-simulator validation deferred to future work. The simulator uses a non-deterministically seeded pseudorandom number generator and produces 10–20% drift in conflict counts between runs, though qualitative outcomes are invariant. The four attack types are scripted behaviours rather than adversarial RL, so the results do not yet support claims under reactive adversarial pressure. Five variants demonstrate that the Pareto result is not a single-scenario artifact but do not support statistical claims at the variant level.

8. Conclusion

We presented a 3-way merge framework for branched cybersecurity simulation timelines. The framework contributes a seven-type conflict taxonomy specific to simulation runtime state, a declarative resolution policy engine that expresses analyst preferences as (conflict type, property key, resolution) triples, and a case study evaluating the framework on a scripted multi-stage attack across four experimental scenario variants and one control. Policy-driven merging produced hybrid timelines that Pareto-dominate both source branches on safety-relevant state metrics in all four experimental variants, under the precondition of separable per-property preferences and monotone metrics. The control variant produced zero conflicts and an idempotent merge, validating the detector on degenerate input. Pure conservative merging failed to produce a usable timeline under the same construction, a failure traced to the structural asymmetry between an LCA-local base and post-branch state established formally in 3.3. The empirical and structural results together motivate the policy engine as a principled mechanism for handling the LCA-asymmetry in 3-way merge frameworks operating on simulation runtime state.

Future work can be extended in three directions. First, the case study exercised only the STATE_VALUE conflict type. Scenarios involving dynamic network reconfiguration, agent lifecycle events, or reactive topology rewiring would exercise the ENTITY_EXISTENCE, LIFECYCLE, and STRUCTURAL categories and stress the resolution grammar's full vocabulary. Second, the implementation and evaluation are bound to a single simulator. Cross-simulator validation against CybORG CAGE and YAWNING TITAN would establish whether the abstract Timeline representation is portable across simulation platforms with different state models. Third, the current implementation seeds its pseudorandom number generator non-deterministically. A seedable generator with deterministic event ordering would support exact reproducibility and enable statistical claims at the variant level.

Acknowledgments

This research was funded by the Slovenian Research Agency, Research Core Funding No. P2-0057–Information systems.

Declaration on Generative AI

During the preparation of this work, the author(s) used Claude (Anthropic) to check grammar and spelling, improve the readability of complex paragraphs, and assist with debugging and auxiliary experiment log files. The design and methodology of the work are the author(s)' own. After using these tools, the author(s) reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] S. Vyas, V. Mavroudis, P. Burnap, Towards the deployment of realistic autonomous cyber network defence: a systematic review, ACM Comput. Surv. 58(1), Article 5 (2025). https://doi.org/10.1145/3729213

[2] D. Allison, P. Smith, K. McLaughlin, Digital twin-enhanced incident response for cyber-physical systems, Proc. ARES 2023, Article 28 (2023). https://doi.org/10.1145/3600160.3600195

[3] J. Pearl, Causality. Models, Reasoning, and Inference, 2nd ed., Cambridge University Press (2009).

[4] T. Mens, A state-of-the-art survey on software merging, IEEE Trans. Softw. Eng. 28(5), 449–462 (2002). https://doi.org/10.1109/TSE.2002.1000449

[5] M. Hybinette, R. M. Fujimoto, Cloning parallel simulations, ACM Trans. Model. Comput. Simul. 11(4), 378–407 (2001). https://doi.org/10.1145/508366.508370

[6] DoltHub: Dolt. Git for data, https://www.dolthub.com/ (accessed May 2026).

[7] PTC Onshape: Branching and merging in Onshape, https://www.onshape.com/en/features/branch-merge-cad (accessed May 2026).
[8] C. Brun, A. Pierantonio, Model differences in the Eclipse Modeling Framework, UPGRADE 9(2), 29–34 (2008).
[9] K. Cheng, P. Cuvin, A. Olechowski, S. Zhou, User perspectives on branching in computer-aided design, Proc. ACM Hum.-Comput. Interact. 7 (CSCW2), Article 371, pp. 1–30 (2023). https://doi.org/10.1145/3610220
[10] M. Sharbaf, B. Zamani, G. Sunyé, Conflict management techniques for model merging. A systematic mapping review, Softw. Syst. Model. 22(3), 1031–1079 (2023). https://doi.org/10.1007/s10270-022-01050-9
[11] M. Standen, M. Lucas, D. Bowman, T. J. Richer, J. Kim, D. Marriott, CybORG: A Gym for the Development of Autonomous Cyber Agents, arXiv preprint arXiv:2108.09118 (2021). https://doi.org/10.48550/arXiv.2108.09118
[12] M. Kiely, M. Ahiskali, E. Borde, B. Bowman, D. Bowman, D. van Bruggen, et al., CAGE Challenge 4: A scalable multi-agent reinforcement learning gym for autonomous cyber defence, AI Magazine 46 (3), e70021 (2025). https://doi.org/10.1002/aaai.70021
[13] K. Hammar, N. Dhir, R. Stadler, Optimal Defender Strategies for CAGE-2 Using Causal Modeling and Tree Search, arXiv preprint arXiv:2407.11070 (2024). https://doi.org/10.48550/arXiv.2407.11070
[14] A. Andrew, S. Spillard, J. Collyer, N. Dhir, Developing Optimal Causal Cyber-Defence Agents via Cyber Security Simulation, ICML 2022 Workshop on Machine Learning for Cybersecurity (ML4Cyber), arXiv preprint arXiv:2207.12355 (2022). https://doi.org/10.48550/arXiv.2207.12355
[15] B. Gyevnár, C. G. Lucas, S. V. Albrecht, S. B. Cohen, Integrating Counterfactual Simulations with Language Models for Explaining Multi-Agent Behaviour, Proc. AAMAS 2026 (to appear), arXiv:2505.17801 (2025). https://doi.org/10.48550/arXiv.2505.17801
[16] X. Li, W. Cai, S. J. Turner, Cloning Agent-Based Simulation on GPU, Proc. PADS 2015, pp. 173–182 (2015). https://doi.org/10.1145/2769458.2769470
[17] M. L. Olson, R. Khanna, L. Neal, F. Li, W.-K. Wong, Counterfactual State Explanations for Reinforcement Learning Agents via Generative Deep Learning, Artif. Intell. 295, 103455 (2021). https://doi.org/10.1016/j.artint.2021.103455
[18] R. O'Driscoll, C. Hagen, J. Bater, J. Adams, Multi Objective Reinforcement Learning for Automated Resilient Cyber Defence, Appl. AI Lett. 6 (3), e70007 (2025). https://doi.org/10.1002/ail2.70007