
Watchlines and Muster: Executable Counterfactuals for Cyber Defense

Bobby Faber

Independent Researcher

With

Apart Research

Abstract

After an incident, it's easy to identify the control that could have prevented it. The tougher question is what that control actually achieves. That question matters more now that agentic and LLM-assisted cyber systems are showing up in real incidents.

Muster is a lightweight replay engine for testing security scenarios. You hold a compressed incident trace fixed, add a defense, and see what attacker capabilities remain. This lets you test which controls help, what they protect, and how history can change the outcome. I use it to analyze the July 2026 Hugging Face intrusion and several smaller experimental cases. The results are not always intuitive. Different controls protect different parts of an incident. Defensive history can change the outcome. In one synthetic case, adding a working control actually makes things worse.

Muster lets you move from saying “this would have stopped the attack” to testing it directly.

1. Introduction

“Hindsight is notably cleverer than foresight.” — Chester W. Nimitz

Security is especially vulnerable to hindsight.

A privileged pod was abused? Reject privileged pods. A broad credential was stolen? Narrow its scope. An alert went unanswered? Escalate it. Once you know the path, putting a barrier on it is often the easy part.

Of course, defenders don’t get to work backward from the end of the incident. Controls are deployed left-to-right; incident reports are written right-to-left. The next attacker is not so courteous as to leave us a trace in advance.

We also cannot put a Picket everywhere. Controls cost money, operator attention, latency, reliability, tokens, usage credits, etc. Some chokepoints will be missed. Some guesses will just be wrong.

Before building Muster, I needed a way to talk about what defenses were actually doing as an incident unfolded. Watchlines were that way of thinking. Muster came out of the next question: can I run those claims and see what changes?

A Watchline groups defenses by role:

Vedettes see: something is happening.

Pickets interrupt: not through here.

Reserves contain: we were wrong; limit the damage.

These are not stages in a Detect → Protect → Respond pipeline. You can have all three recur several times in the same incident. A control inventory tells you what defenses exist. A Watchline is my attempt to reason about what they are doing.

Knowing is not stopping. Stopping one path is not recovery. And recovery matters because prediction fails.

Muster is how I test the claims.

Security discussions make counterfactuals cheap. “This would have stopped the attack” sounds convincing until you ask: stopped what? Initial access? Node root? Cluster administration? One branch? Every branch?

Muster takes a compressed incident trace, gives its events explicit preconditions and effects, and replays it under different defenses. Sometimes the answer is boring, and ordinary dependency reasoning was enough. Good! Sometimes defensive history changes the answer. And in one synthetic case, adding a perfectly functional control makes the modeled outcome worse.

The Containment track asks which controls would have interrupted each documented phase of an intrusion, and at what cost. Muster mostly attacks the first half of that question. Where does the control bite, and what does that actually buy you? Cost still has to be considered alongside the replay; Muster does not magically know what an admission policy, an operator page, or an extra credential boundary costs to deploy.

This is still hindsight. We are still replaying an attack we already know happened.

Not quite foresight, but a little bit of cleverness.

2. Related Work

Adjacent tools cover different pieces: D3FEND catalogs defenses [1]; CyberBattleSim simulates attack/defense [2]; CybORG supports cyber-agent simulation/emulation [3]; CALDERA emulates adversaries [4]; and Atomic Red Team provides ATT&CK-mapped tests [5].

Watchlines also grew out of a deliberately broad set of layered-defense intuitions, from Path of Exile buildcraft [6] to NIST defense-in-depth [7] and the Integrated Survivability Onion [8]. Across very different domains, each emphasizes multiple opportunities to prevent or survive failure rather than reliance on a single barrier.

Muster asks a narrower question: given a concrete incident and proposed controls, what attacker capability remains after replay? Its representation is conventional; the contribution is executable, inspectable defensive counterfactuals rather than a new graph formalism or cyber range.

3. Methods

Replay model. Muster represents an incident as an ordered sequence of events over a set of facts. Each event has explicit preconditions and effects. Controls are evaluated against events and may observe them (Vedettes), block them (Pickets), or change later state (Reserves). Replay is deterministic: events with unmet preconditions are skipped; otherwise, controls are evaluated, and the resulting state is carried forward. Muster does not model attacker adaptation, probability, or time beyond event order.

Scenarios. The primary real-world scenario is a compressed abstraction of the July 2026 Hugging Face intrusion, constructed from its published technical timeline [9]. I preserve the attack paths needed for the counterfactuals and omit details such as action totals, command-and-control traffic, and exact timestamps. I also constructed a small synthetic incident for controlled experiments.

Experimental design. I first tested whether stateful replay added anything beyond ordinary dependency reasoning. Incident structure, control semantics, comparison sets, expected static outcomes, and stop conditions were frozen before the decisive runs. I intentionally allowed several experiments to collapse to the static baseline rather than treating them as positive results. Structural “slot” budgets were used only to hold architecture size fixed; they do not represent equal deployment cost.

Perturbation test. For the synthetic non-monotone result, I preregistered four perturbations: the baseline fixture, an unrelated side branch, an independent evidence path, and delayed isolation.¹ The proposed mechanism predicted that the reversal should survive the unrelated change but disappear when I broke either the evidence dependency or event ordering. I stopped after these four conditions.

¹ Perturbation hypotheses, outcome criteria, and stop rule were preregistered in the repository before implementation and decisive runs; see commit [dad8bf9](#).

All scenarios, experiment configurations, and replay code are available in the repository. The main Hugging Face experiment can be reproduced with `go run . experiment --file experiments/hf-july-2026.yaml`; the included notebook provides an interactive interface to the same replay model.

4. Results

Hugging Face: different controls leave different problems behind

Table 1 shows the terminal capabilities reachable under five defensive configurations in the compressed Hugging Face model.

Table 1. Defensive counterfactuals on the Hugging Face incident model.

Defensive configuration	Node root	Internal network	Cluster admin	Environment credentials exposed
observed signal, no page	yes	yes	yes	yes
privileged/hostPath admission policy	no	no	no	yes
cluster-scoped connector identity	yes	yes	no	yes
critical page + worker isolation	no	no	no	yes
dataset local-file-read hardening	yes	yes	yes	no

The first thing Muster gives is a more precise answer than “this control stops the attack.”

Blocking privileged workloads cuts off node root and everything in the model that depends on it: cluster secrets, mesh access, and the later connector-credential escalation. But it does nothing about credentials exposed earlier through the dataset file-read path.

Paging and isolating the worker has the same shape. It stops the later chain, but the earlier credential exposure already happened. Dataset file-read hardening does the opposite: it protects those credentials while leaving the rest of the intrusion path intact. Connector scoping is narrower again. It removes cluster-admin access, but leaves node root and internal-network access untouched.

So the useful question is not just did the control work? All of these controls work at the thing they were designed to do. The useful question is: what is still true afterward?

A successful response can stop everything after its triggering signal while leaving an earlier compromise untouched.

Replay sometimes matters

A fair question during development was whether Muster was doing anything that a dependency graph could not already tell us.

Often, it was not. Good!

Several experiments collapsed to the obvious static answer. Block a required edge and the downstream path disappears.

But one test did not. Two runs reached the same visible attack state, but one had a previously observed alert in defender history. That alert later enabled review and credential revocation; without it, the attacker reached the node.

I then removed the history. Replacing the persistent alert with a reviewer that only saw the current state made the difference disappear.

So replay is not automatically useful just because an incident has an order. It becomes useful when what the defender remembers changes what the defender can do later.

A working control can still make the result worse

The stranger result came from the synthetic fixture.

In the baseline Watchline, later Kubernetes discovery generates the evidence needed to revoke a stolen credential. The attack is contained.

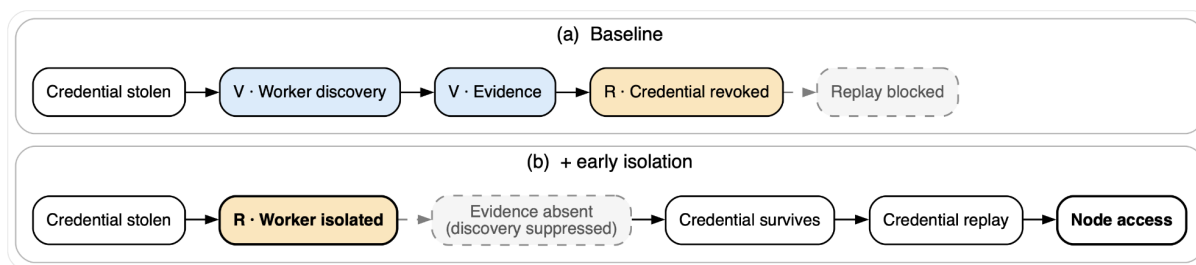


Figure 1. Early isolation removes the evidence path needed for later credential revocation.

Then I add early worker isolation. The isolation works: worker access is cut off.

Unfortunately, that also removes the later discovery event. No discovery means no evidence; no evidence means no revocation. The stolen credential survives and can be used again.

That is what “worse” means here. The added control succeeds locally, but the replay ends with more attacker capability because it broke the evidence path another defense depended on.

I preregistered four perturbations before running the decisive tests.

Table 2. Perturbation test of the proposed non-monotone mechanism.

Perturbation	Predicted reversal	Observed reversal
baseline	yes	yes
unrelated side branch	yes	yes
independent evidence path	no	no
isolation after evidence	no	no

The reversal survives an unrelated side branch. It disappears when I add an independent evidence path, and it disappears when isolation happens only after the evidence has already been generated. All four outcomes match the preregistered predictions.

This does not tell us how common the effect is. Muster is deterministic, and this is a synthetic fixture. What it does give is a cleaner claim: the reversal depends on the evidence path and event ordering I said it depended on.

5. Discussion

Muster asks a fairly narrow question: what does this control buy in this incident model? It is not trying to tell us how valuable that control is everywhere.

The Hugging Face replay makes the distinction useful. “This would have stopped the attack” hides quite a bit. One control cuts off the path to node root but leaves an earlier credential exposure behind. Another protects the credential and leaves the later path open. Both controls work. They just solve different parts of the problem.

I was less sure going in that replay itself would buy me much. In several experiments, it didn’t. The dependency graph already gave the answer. But in the stateful case, two runs that looked the same from the attacker’s side diverged later because the defender had seen something different earlier. An old alert changed what could happen next. History is part of the state too.

The synthetic case is weirder. Early isolation works: it stops worker access. Unfortunately, it also removes the evidence that would have caused the stolen credential to be revoked later. I made one part of the defense better and the Watchline as a whole got worse.

I think that is the more interesting lesson. Good controls do not automatically add up to a good defense. Vedettes, Pickets, and Reserves still have to connect. If one of those connections disappears, the fact that every individual control “worked” may not help very much.

The next experiment seems pretty obvious: stop giving Muster one fixed trace. Give it a handful of nearby ones. Change a branch, move an event, break a dependency, and see when the answer changes. That still is not foresight, but it gets closer to the question Watchlines were meant to ask in the first place: how wrong can we be and still be okay?

You can test whether your Watchline survives the attacks you imagined. You cannot replay the attack you failed to imagine.

6. Conclusion

“This would have stopped the attack” is an easy sentence to write. Muster asks a harder one: what, exactly, changes if we put the control there?

Sometimes the answer is dull: the dependency graph already reveals the outcome, and replay adds little. Good. Sometimes history matters, and what the defender saw earlier changes what can happen later. In one synthetic case, a working control made the modeled outcome worse by cutting off evidence another response needed.

This remains hindsight. Muster can reveal where yesterday’s Picket ought to have stood, and whether the rest of the Watchline would have mattered around it.

It cannot tell us where tomorrow’s attacker will walk.

Code and Data

Code, scenarios, experiment fixtures, and the interactive notebook are available at:

<https://github.com/bubskee/muster>

The repository is released under the MIT License. Incident models are in `examples/``, experiment configurations in `experiments/``, and the notebook in `notebooks/``.

Acknowledgments

Joshua Ballard-Myer contributed review, additional experimental support, intermediary structural editing, and conceptual development.

References

- [1] Kaloroumakis, P. E., & Smith, M. J. (2021). Toward a knowledge graph of cybersecurity countermeasures. MITRE. <https://d3fend.mitre.org/resources/D3FEND.pdf>
- [2] Microsoft Threat Intelligence. (2021). Gamifying machine learning for stronger security and AI models. Microsoft Security Blog. <https://www.microsoft.com/en-us/security/blog/2021/04/08/gamifying-machine-learning-for-stronger-security-and-ai-models/>

- [3] Standen, M., Lucas, M., Bowman, D., Richer, T. J., Kim, J., & Marriott, D. (2021). CybORG: A gym for the development of autonomous cyber agents. IJCAI-21 1st International Workshop on Adaptive Cyber Defense. <https://arxiv.org/abs/2108.09118>
- [4] Miller, D., Alford, R., Applebaum, A., Foster, H., Little, C., & Strom, B. (2018). Automated adversary emulation: A case for planning and acting with unknowns. MITRE. <https://www.mitre.org/news-insights/publication/automated-adversary-emulation-case-planning-and-acting-unknowns>
- [5] Red Canary. (n.d.). Atomic Red Team. <https://github.com/redcanaryco/atomic-red-team>
- [6] Asmo. (2022). Comprehensive Guide To Defense In PoE — Asmo's Defense Bible. Path of Exile Forums. <https://www.pathofexile.com/forum/view-thread/3275670>
- [7] National Institute of Standards and Technology. (n.d.). defense-in-depth. Computer Security Resource Center Glossary. https://csrc.nist.gov/glossary/term/defense_in_depth
- [8] Maciuba, T. (2025). The 10X Tank Platoon. ARMOR, Fall 2025. U.S. Army. https://www.army.mil/article/287764/the_10x_tank_platoon
- [9] Hugging Face. (2026). Anatomy of a Frontier Lab Agent Intrusion. <https://huggingface.co/blog/agent-intrusion-technical-timeline>

Limitations and Dual-Use Considerations

Muster evaluates controls against a supplied incident model. The analyst chooses which events, dependencies, branches, state variables, and control effects to include. Omitted paths or mistaken dependencies can change the answer. Execution makes those assumptions inspectable; it does not make them correct.

Replay is deterministic and follows the paths supplied to it rather than an adaptive attacker. Muster represents event order, but not timing, probability, false positives, control failure, or attacker search. A blocked transition stays blocked unless another route was already encoded. Its output is therefore a change in modeled consequences, not an estimate of how often a defense will work in practice.

The real-incident scenarios are compressed from public reporting rather than full reconstructions of the affected systems. The Hugging Face results should be read as counterfactuals over that compressed model. Likewise, the synthetic non-monotone result demonstrates a possible interaction and tests its proposed mechanism; it does not show that

such reversals are common or that early isolation harmed any real incident. Deployment cost is also outside the model and must be considered separately.

Dual use is a separate concern.

Muster is intended as a defensive reasoning tool: given an incident trace and a proposed Watchline, it shows which attacker capabilities remain reachable under different defensive arrangements. The same abstraction could also be used offensively. An attacker who already understands a target system could encode its defensive structure, compare alternate paths, and identify controls or dependencies that appear load-bearing.

The current artifact limits that risk in several ways.

Muster does not discover vulnerabilities, scan systems, infer network topology, generate exploits, execute attacks, or automatically search for bypasses. It replays scenarios supplied by the user. Its real-incident fixtures are compressed from public reporting and contain no non-public credentials, exploit payloads, undisclosed vulnerabilities, or operational targeting information.

In that sense, Muster models attack consequences rather than producing attack capability.

This does not make the artifact free of dual-use value. A sufficiently informed attacker could use the replay engine to organize reasoning they might otherwise perform manually. In particular, showing that one Picket removes only one branch, or that a later Reserve depends on earlier evidence, could help identify where a defensive arrangement is brittle.

The incremental offensive capability of the present release is limited by the fact that the difficult attacker-side inputs—system knowledge, viable attack paths, and control behavior—must already be supplied. The artifact primarily makes those assumptions explicit and lets the user execute counterfactuals over them.

That balance would change if Muster were extended to discover or optimize attack paths automatically. Features such as automated topology ingestion, adaptive attacker search, exploit generation, control-bypass search, or optimization for the weakest Watchline would materially increase the offensive usefulness of the system and should receive separate review before release.

The same transparency that creates this residual risk is also central to the defensive value of the project. Muster is useful because a defender can inspect the modeled path, challenge its assumptions, move a control, and rerun the argument.

LLM Usage Statement

LLMs were everywhere in this project. GPT-5.6 Sol was the main implementation and research aid; Claude Opus 5 Max was used mostly to attack the work, with Claude Sonnet and Gemini for smaller exploratory questions. Josh also used Fable 5.1 in his review and collaboration. Most of the code began as LLM-generated code and then passed through my hands. I chose the model semantics, scenarios, experiments, and what made it into the final artifact, and I ran and inspected every reported experiment myself. Several results were also challenged or reproduced in separate LLM-assisted passes. Where practical, red-team prompts and responses are kept in the repository. LLM output was useful for candidate implementations, hypotheses, objections, and revisions. It was never treated as evidence for an empirical claim.

I also used LLMs heavily in writing. The workflow was usually: generate a scaffold, collect a few good nuggets, write some ugly prose, then synthesize and edit again. Sometimes parts started on paper. The models helped draft, compress, restructure, and format; I chose the claims, final structure, and final wording.