The Temporal Attack Surface: A Δt Framework for Asynchronous Security Systems

James Beck
Independent Researcher

Abstract

Security systems with temporally decoupled detection, decision, and response exhibit a **common structural vulnerability class**. Building on the Δt framework for temporal dynamics in hierarchical systems and its recent application to Deep Packet Inspection circumvention, we demonstrate that SIEM alerting, CI/CD gates, authentication flows, rate limiting, and human-in-the-loop approvals share failure modes when fast commitment outruns slow grounding. We formalize the Temporal Attack Surface as the gap between evidence accumulation (slow) and enforcement action (fast), and derive conditions under which asynchronous security systems fail. This framework provides defender-side instrumentation to identify race windows, audit commitment polarity (fail-open vs. fail-closed defaults), and measure the temporal gap between detection and response. We show that "zero trust" architectures implicitly minimize temporal coupling, and that human approval processes represent the slowest—and therefore most vulnerable—layer in security chains. The unifying insight: security failures in asynchronous systems are control-theoretic race conditions, not configuration errors.

Keywords: temporal attack surface, Δt framework, asynchronous security, race conditions, SIEM, CI/CD security, authentication, defender instrumentation

---

1. Introduction

Security systems are increasingly asynchronous: detection happens in one process, decision-making in another, and enforcement in a third. This temporal decoupling—necessary for performance, scalability, and human oversight—creates a fundamental vulnerability: **the gap between observation and action**.

Recent work [Beck 2025] demonstrated that Deep Packet Inspection (DPI) circumvention exploits this gap systematically. ClientHello fragmentation, protocol obfuscation, and Encrypted ClientHello (ECH) all manipulate the relationship between when evidence arrives (transport clock) and when the censor can act (inspection clock). The key insight: **bypass occurs when evidence accumulation cannot keep pace with the commitment deadline**.

This paper generalizes that insight. We show that:

1. **All asynchronous security systems** share the same temporal structure
2. **The Universal Bypass Inequality (UBI)** from DPI analysis applies across domains
3. **Defender instrumentation** can identify and measure temporal vulnerabilities
4. **Mitigation strategies** follow directly from understanding temporal coupling

The Δt framework—previously applied to institutional failures, LLM hallucinations, and organizational dynamics—provides the theoretical foundation. The contribution of this work is demonstrating its applicability to **security-as-control-problem**: where fast attackers race slow defenders for commitment authority.

1.1 The Core Problem

Consider any security system with three stages:

Detection: Evidence accumulates (logs, signals, patterns)
Decision: Policy evaluated (rules, heuristics, human judgment)
Response: Action taken (block, alert, escalate)

These stages operate at **different speeds**: - Detection: milliseconds to hours (depends on sampling rate, aggregation windows) - Decision: microseconds to days (depends on automation vs. human-in-loop) - Response: milliseconds to weeks (depends on automated enforcement vs. manual remediation)

An attacker wins by ensuring:

T_objective < min(T_detect, T_decide, T_respond)

That is: complete the objective before any stage can prevent it.

This isn't exploitation of bugs or misconfigurations—it's **structural exploitation of temporal gaps**.

1.2 Why This Matters

Traditional security thinking focuses on: - Access control (who can do what) - Vulnerability management (fixing bugs) - Configuration hardening (closing gaps)

The temporal attack surface is **orthogonal** to these. A perfectly configured system with no vulnerabilities still fails if: - Detection is too slow - Decisions arrive too late - Response happens after commitment

"Defense in depth" becomes "layers of asynchronous observers" each with their own temporal constraints. The attacker simply needs to be **faster than the slowest observer**.

1.3 Contributions

Positioning: Current security models (attack trees [14], kill chains [15], MITRE ATT&CK [16]) are **static**—they map prerequisites and techniques but not **temporal exploitability**. Time-of-check to time-of-use (TOCTOU) races [17] and asynchronous intrusion detection [18] touch on temporal aspects but treat them as implementation bugs rather than fundamental control-theoretic properties. This work provides the **dynamic, temporal complement** to static security models.

Contributions:

1. **Formalize the Temporal Attack Surface** as detection-decision-response temporal gap
2. **Demonstrate universality** across five security domains (SIEM, CI/CD, auth, rate limiting, human-in-loop)
3. **Derive defender instrumentation** to measure temporal vulnerabilities in deployed systems
4. **Provide mitigation framework** based on temporal coupling principles

---

2. Background: The Δt Framework

The Δt (delta-time) framework analyzes failures in hierarchical systems where temporal layers operate at mismatched speeds [Beck 2025]. Core insight: **systems fail when fast layers outrun slow layers**.

2.1 Temporal Coherence

A system maintains **temporal coherence** when:

Commitment rate ≤ Grounding rate

That is: the rate at which the system makes decisions does not exceed the rate at which it can verify those decisions against reality.

Violations manifest as: - Institutions making decisions faster than evidence accumulates - LLMs generating claims faster than grounding can verify - Organizations committing resources faster than feedback arrives - **Security systems enforcing policy faster than threats can be detected**

2.2 The Universal Bypass Inequality (UBI)

From DPI analysis [Beck 2025], a bypass exists when:

(T_E > W_j) ∨ (Cost(E) > B_j) ∨ (E(t) = ∅)

∧ T_E + A_j ≥ T_commit

∧ T_handshake < U

Where: - **T_E**: Time until decisive evidence exists - **W_j**: Observation window (how long defender waits for evidence) - **Cost(E)**: Computational cost to process evidence - **B_j**: Processing budget per observation - **E(t)**: Evidence available at time t - **A_j**: Action latency (time to enforce decision) - **T_commit**: When attacker objective completes - **U**: Patience threshold (when attacker gives up)

Interpretation: Attacker succeeds if evidence arrives too late, costs too much to process, or doesn't exist—AND enforcement happens after commitment—AND attacker doesn't timeout.

This paper demonstrates the UBI generalizes beyond DPI to **all asynchronous security**.

---

3. The Temporal Attack Surface Model

We define the **Temporal Attack Surface** as the set of race windows in a security system where attacker commitment can outrun defender response.

3.1 Three Clocks

Every asynchronous security system has at minimum three clocks:

Detection Clock (T_detect): - When evidence accumulates to decision threshold - Depends on: sampling rate, aggregation window, correlation depth - **Vulnerable when**: Evidence fragments across sampling windows, stays below detection threshold, or exploits correlation depth limits

Decision Clock (T_decide): - When policy evaluation completes - Depends on: rule complexity, human latency, approval chains - **Vulnerable when:** Timeout defaults trigger, approval fatigue occurs, or edge cases exhaust evaluation budget

Response Clock (T_respond): - When enforcement action takes effect - Depends on: automation latency, manual intervention, remediation complexity - **Vulnerable when:** Objective completes before response activates, or response requires costly manual intervention

3.2 Commitment Points

The critical metric is **T_commit**: when the attacker's objective becomes irreversible.

Examples: - **Data exfiltration**: When data leaves the network boundary - **Privilege escalation**: When elevated session is granted - **Code deployment:** When artifact is signed/promoted - **Account takeover:** When session token is issued

Security systems fail when:

T_commit < min(T_detect, T_decide, T_respond)

3.3 Commitment Polarity (C_j)

Every decision point has a **default under uncertainty**:

Fail-open (C_j = allow): - Allow action when evidence insufficient or timeout exceeded - Optimizes for availability, user experience - **Vulnerable to:** Δt inflation (make detection slow), cost exhaustion (make processing expensive)

Fail-closed (C_j = deny): - Deny action when evidence insufficient or timeout exceeded - Optimizes for security, risk aversion - **Vulnerable to:** False positives (legitimate traffic blocked), availability impact

The attacker's goal: **force the system into its default state** where uncertainty exists.

3.4 Political Budget (κ_j)

Every security control has a **tolerance for disruption**:

κ_j: Maximum acceptable cost of false positives, service disruption, or operational overhead

When **Enforcement_Cost > κ_j**, the control is relaxed, disabled, or circumvented by users/operators.

Examples: - CI/CD gates with "skip on timeout" because blocking releases is too costly - SIEM alerts tuned down because SOC is overwhelmed - MFA disabled because users complain - Rate limits loosened because legitimate traffic is blocked

Key insight: Attackers don't need to defeat controls—they need to make controls **politically untenable**.

---

4. Case Studies

We demonstrate the Temporal Attack Surface across five security domains.

4.1 SIEM / Security Operations Center (SOC)

System structure: - Logs generated at endpoints/network devices - SIEM aggregates and correlates logs - Analysts (or SOAR) evaluate alerts - Response team takes action

Temporal parameters:

W_j (Detection Window): Mean Time to Detect (MTTD) - Industry average: 200+ days for breaches - Real-time alerting: seconds to minutes - Batch correlation: hours to days

T_decide: Analyst triage time - Automated: milliseconds - Human: minutes to hours (depends on alert queue)

A_j (Response latency): Mean Time to Respond (MTTR) - Automated isolation: seconds - Manual remediation: hours to days

C_j (Commitment polarity): - Alert-only: Fail-open (log and continue) - Auto-isolate: Fail-closed (block on suspicion)

T_commit (Attacker objectives): - Data exfiltration: 10MB at 1Mbps = 80 seconds - Lateral movement: Varies (often < 1 hour) - Persistence: Varies (often < 1 day)

Temporal attack surface:

If **T_commit < W_j** (objective completes before detection), attacker wins regardless of subsequent response.

Low-and-slow: Keep evidence accumulation below detection threshold by: - Spreading activity over time (< W_j per window) - Staying below anomaly thresholds (Cost(E) < B_j for classification) - Mimicking legitimate patterns (reduce E(t) score)

High-and-fast: Complete objective before detection by: - Burst all activity into single window (< W_j total) - Accept that detection will occur but response arrives late (T_commit < T_detect + A_j)

Defender instrumentation:

1. **Measure MTTD distribution** (not just average)
   - What's 95th percentile? 99th percentile?
   - Which detection types are slowest?
2. **Map T_commit for critical assets**
   - How long to exfiltrate 10GB? 100GB? 1TB?
   - How long to establish persistence?
   - How long to escalate privileges?
3. **Audit C_j defaults**
   - Which alerts are log-only (fail-open)?
   - Which trigger auto-isolation (fail-closed)?
   - What's the false positive rate on each?
4. **Measure κ_j (political budget)**
   - At what false positive rate do analysts disable alerts?
   - What's the queue depth before triage becomes meaningless?

Mitigation strategies:

- **Reduce W_j:** Real-time streaming analytics, faster correlation
- **Reduce T_commit:** Rate limit data egress, privilege escalation controls
- **Increase A_j cost:** Automated response (move decision closer to detection)
- **Adjust C_j:** Fail-closed for critical assets, accept availability trade-off

4.2 CI/CD Security Gates

System structure: - Code committed to repository - Pipeline runs: build, test, security scan (SAST/DAST) - Approval gates (manual or automated) - Artifact signed and deployed

Temporal parameters:

W_j (Gate timeout): - Per-stage timeout: 5-30 minutes typical - Queue depth: Can add hours to days under load - **Critical:** Many pipelines configured "skip on timeout" (fail-open)

T_decide: - SAST/DAST scan time: minutes to hours - Manual approval: hours to days - **Bottleneck:** Human approvals are slowest layer

A_j (Block latency): - Automated: Immediate (break build) - Manual: Requires revert/rollback (slow, costly)

C_j (Commitment polarity): - **Fail-open:** Skip scan on timeout, advisory-only findings - **Fail-closed:** Break build on timeout, block on any finding

T_commit: - **Pre-merge:** Code in feature branch (low commit) - **Post-merge:** Code in main branch (medium commit) - **Post-deploy:** Code running in production (high commit) - **Post-sign:** Artifact cryptographically signed (irreversible commit)

Temporal attack surface:

Commitment boundary misplacement: The core CI/CD temporal failure occurs when **commitment boundaries are placed upstream of security certainty**: - **Merge** is irreversible socially (team expects it in main) - **Deploy** is irreversible operationally (running in production) - **Signing** is irreversible cryptographically (artifact trusted)

Critical question: Which security gates occur **before** each commitment boundary?

If signing occurs before SAST completes, or deployment happens before manual approval, the temporal race window exists regardless of specific implementation.

Implementation-specific example: Dependency resolution that runs after SAST but before signing creates a desynchronization where malicious packages enter without inspection.

T_E > W_j: Large or complex changes that cause scanner timeout - Pipeline configured to "skip on timeout" (fail-open) - Code merges without security review

Cost(E) > B_j: Exploit expensive analysis paths - Nested dependencies - Obfuscated code patterns - Large codebases that exhaust scan budget

Defender instrumentation:

1. **Map T_commit boundaries**
   - Where is the point of no return?
   - Is signing before or after all gates?
   - Can deployment be reverted cheaply?
2. **Audit fail-open configurations**
   - Which gates skip on timeout?
   - Which findings are advisory-only?
   - What's the timeout threshold?
3. **Measure queue depth dynamics**
   - What's average time-in-queue?
   - Does queue depth vary (daily/weekly patterns)?

- Can attackers time submissions for high-load periods?

4. **Identify human bottlenecks**
   - Which approvals require humans?
   - What's the approval latency distribution?
   - Can approval be bypassed under time pressure?

Mitigation strategies:

- **Move T_commit later:** Sign after all gates, not before
- **Reduce W_j:** Faster scanners, incremental analysis
- **Fail-closed:** Break build on timeout for critical pipelines
- **Remove human gates:** Automate approval for low-risk changes
- **Provenance:** SLSA-style supply chain verification

4.3 Authentication & Access Control

System structure: - User attempts login - Credentials verified - Policy evaluated (MFA, device trust, geo-location) - Session granted or denied

Temporal parameters:

W_j (Lockout window): - Failed attempt counter: 5-10 attempts typical - Time window: 5-15 minutes typical - Reset after lockout: 15 minutes to 24 hours

T_decide: - Credential check: Milliseconds - MFA prompt: Seconds to minutes (user interaction) - Risk-based auth: Milliseconds (automated) to minutes (human review)

A_j (Block latency): - Automated lockout: Immediate - Manual investigation: Hours to days

C_j (Commitment polarity): - **Fail-open:** Lockout per-IP (attacker can rotate IPs) - **Fail-open:** CAPTCHA instead of hard block - **Fail-closed:** Account lockout (affects legitimate user)

T_commit: - Session token issued (irreversible without revocation)

Temporal attack surface:

Credential stuffing: Distribute attempts across IPs/time to stay below W_j threshold per source.

MFA fatigue: Spam MFA prompts until user approves (exploit human A_j).

T_E > W_j: Complete brute force within single lockout window: - 10 attempts in 5 minutes - If successful, session granted before lockout - Lockout happens but attacker already has access

Defender instrumentation:

1. **Measure W_j effectiveness**
   - What's actual lockout threshold?
   - Can attackers stay below it?
   - Does lockout reset allow retry?
2. **Map T_commit**
   - When is session token issued (before or after all checks)?
   - Can token be revoked quickly?
   - What's the token lifetime?
3. **Human response latency**
   - How long before users report suspicious MFA prompts?
   - How long before manual investigation begins?
4. **C_j analysis**
   - What happens on timeout (user doesn't respond to MFA)?
   - What happens on uncertain risk scores?

Mitigation strategies:

- **Reduce W_j:** Lower attempt threshold, shorter window
- **Adaptive W_j:** Decrease threshold based on risk signals
- **Fail-closed:** Account lockout on suspicious patterns (accept usability hit)
- **Increase A_j cost:** Require re-auth for sensitive actions even with valid session

4.4 Rate Limiting & Abuse Prevention

System structure: - Requests arrive at API/service - Rate limiter checks: requests per window, burst capacity - Policy applied: allow, throttle, or deny - Response returned

Temporal parameters:

W_j (Rate window): - Token bucket: Refill rate (e.g., 100 requests/second) - Sliding window: Time span (e.g., 1000 requests/hour)

T_decide: - Automated: Microseconds

A_j (Enforcement): - Immediate: Request denied or queued

C_j (Commitment polarity): - **Fail-open:** Degraded service (slower response) instead of hard deny - **Fail-closed:** Hard deny on limit exceeded

T_commit: - **Per-request:** When request is processed - **Aggregate:** When attacker objective completes (e.g., scrape entire database)

Temporal attack surface:

Burst attack: Use entire token bucket in single burst: - If bucket capacity = 1000 requests - And objective = scrape 1000 records - Complete in single burst before refill

Distributed attack: Spread requests across IPs/accounts to avoid per-source limits.

T_E > W_j: Exploit slow detection of aggregate abuse: - Each source stays under limit - Aggregate abuse undetected until after objective complete

Defender instrumentation:

1. **Measure burst capacity**
   - What's maximum burst size?
   - Can attacker complete objective in one burst?
2. **Aggregate detection latency**
   - How long to detect distributed abuse?
   - What's the aggregation window?
3. **C_j under load**
   - Does system degrade (fail-open) under attack?
   - At what load does it switch to fail-closed?

Mitigation strategies:

- **Reduce burst capacity:** Lower token bucket size
- **Adaptive limits:** Decrease limits under attack
- **Fail-closed:** Hard deny instead of degradation
- **Aggregate limits:** Cross-account/cross-IP rate limiting

4.5 Human-in-the-Loop Approvals

System structure: - Automated system requires human approval - Request queued for human review - Human evaluates request (with varying context/attention) - Approval or denial issued

Temporal parameters:

W_j (Approval latency): - Synchronous: Minutes (human waiting) - Asynchronous: Hours to days (queue depth dependent)

T_decide: - Human cognitive load: Seconds per approval - But: Fatigue, context-switching, queue pressure

A_j (Enforcement): - Immediate after approval

C_j (Commitment polarity): - **Fail-open (fatigue):** Approve to clear queue - **Fail-closed (paranoid):** Deny everything suspicious (high false positive)

κ_j (Political budget): - Approval fatigue threshold - "Just approve it so I can go home"

T_commit: - When approval granted (human decision is irreversible without detection/revert)

Temporal attack surface:

Approval fatigue: Spam approval requests until human approves just to clear queue.

Timing attacks: Submit requests when: - Human is tired (end of shift) - Queue is deep (high cognitive load) - Context is poor (insufficient information to evaluate)

T_E > W_j: Present insufficient context so human approves quickly rather than investigating.

Defender instrumentation:

1. **Measure approval latency distribution**
   - What's median? 95th percentile?
   - Does it correlate with queue depth?
   - Does it correlate with time-of-day?
2. **Audit approval patterns**
   - Do approval rates increase with queue depth (fatigue)?
   - Do approval rates vary by approver?
   - Are there time-of-day effects?
3. **Measure context quality**
   - How much information do approvers have?
   - Can they access additional context easily?
   - Do they actually review context?
4. **κ_j (breaking point)**
   - At what queue depth do approvals become rubber-stamps?
   - What's the false positive tolerance?

Mitigation strategies:

- **Reduce W_j:** Faster approval process (risk: less thorough)
- **Reduce queue depth:** Rate limit approval requests
- **Improve context:** Provide richer information per request
- **Remove humans:** Automate low-risk approvals
- **Split workload:** Multiple approval tiers based on risk

---

5. Unified Framework

5.1 Mapping To UBI

All five domains map to the Universal Bypass Inequality. We can express the core race condition as a single unifying equation:

Attacker succeeds if:

T_commit(Objective) < T_respond(Defender)

Where:

T_respond = max(W_j, T_decide) + A_j if evidence accumulates

T_respond = ∞ if C_j = fail-open and (T_E > W_j ∨ Cost(E) > B_j ∨ E(t) = ∅)

That is: the defender's response time is the maximum of their detection window and decision time, plus action latency—**unless** the system defaults to fail-open when evidence is insufficient, in which case no response occurs.

Domain-specific parameters:

Domain	W_j	E(t)	C_j	T_commit	A_j
SIEM	MTTD	Log correlation	Alert vs. isolate	Data exfiltration	MTTR
CI/CD	Pipeline timeout	SAST/DAST results	Skip vs. break build	Artifact signed	Revert latency
Auth	Lockout window	Failed attempts	CAPTCHA vs. lockout	Session granted	Account disable
Rate Limit	Rate window	Request patterns	Degrade vs. deny	Request processed	Immediate
Human Approval	Approval latency	Request context	Approve vs. deny	Approval granted	Revocation

Universal condition for attacker success:

(T_E > W_j) ∨ **(Cost(E) > B_j)** ∨ **(E(t) = ∅)**

∧ **T_commit < min(T_detect, T_decide, T_respond)**

∧ **T_objective < U** (attacker patience)

5.2 The Precision-Robustness Trade-off

All asynchronous security systems face the same fundamental trade-off:

Precision (targeted enforcement): - Requires tight temporal coupling (small Δt) - Vulnerable to Δt attacks (inflation, desynchronization) - Low false positive rate - High κ_j requirement (operationally expensive)

Robustness (crude enforcement): - Requires loose temporal coupling (large Δt, or E(t) = ∅) - Resistant to Δt attacks - High false positive rate (collateral damage) - Low κ_j tolerance (politically expensive)

Defenders move along this curve as attackers mature:

SIEM: Real-time alerting (precise, brittle) ⟶ Auto-isolation (crude, robust)

CI/CD: Per-commit scanning (precise) ⟶ Signed artifact verification (crude but robust)

Auth: Per-attempt evaluation (precise) ⟶ Device/IP bans (crude)

Rate Limit: Per-request evaluation (precise) ⟶ IP blocking (crude)

Human: Detailed review (precise, slow) ⟶ Automated approval (crude, fast)

Key insight: Attackers force defenders toward crude enforcement by exploiting temporal gaps. ECH in DPI is the canonical example: by removing evidence entirely (E = ∅), it forces censors from precision DPI to crude infrastructure bans.

5.3 Strategic Implications

Structural vulnerabilities to understand: - Identify slowest observer in security chain - Map objectives that can complete faster than observation - Audit fail-open defaults (most systems optimize for availability) - Measure political budget exhaustion points (where controls become too disruptive)

Defender actions (actionable): - **Minimize Δt:** Reduce gap between detection and response - **Move T_commit later:** Delay irreversible decisions - **Audit C_j:** Know your fail-open defaults - **Measure W_j distribution:** Not just averages, understand tail latencies - **Map temporal attack surface:** Where are your race windows?

---

6. Defender Instrumentation

We provide a framework for auditing temporal attack surface in deployed systems.

6.1 Core Metrics

For each security control, measure:

1. **W_j distribution** (not just average)
   - Median, 95th percentile, 99th percentile detection time
   - Varies by detection type, time of day, load
2. **T_commit location**
   - Where is the point of no return?
   - Can actions be reverted cheaply?
   - What's the reversibility window?
   - **Red team the clock:** Don't just test *if* an adverse event can occur, time *how long* it takes to reach the point of no return. Most organizations don't know how fast they can be compromised.
3. **C_j defaults**
   - What happens under uncertainty?
   - What happens on timeout?
   - What's the fail-open vs. fail-closed policy?
4. **κ_j (political budget)**
   - At what false positive rate do operators disable control?
   - What's the operational overhead tolerance?
   - Where does usability override security?
5. **A_j (response latency)**
   - How fast can you act after detection?
   - What's the automation vs. manual ratio?
   - What's the MTTR distribution?

6.2 Audit Questions By Domain

SIEM/SOC: - What's your MTTD for different threat types? - What's the maximum duration an adverse event could operate before detection? - Which alerts are log-only (fail-open)? - What's your analyst queue depth? - At what queue depth does alert triage degrade?

CI/CD: - Where is your T_commit (merge, deploy, or sign)? - Which gates skip on timeout? - What's your pipeline queue depth? - How long do security scans take? - Which approvals require humans?

Authentication: - What's your lockout threshold and window? - What's the maximum credential attempts within a single window? - How long before suspicious MFA prompts are investigated? - What happens on MFA timeout?

Rate Limiting: - What's your burst capacity? - What's the maximum burst size before enforcement? - How long to detect distributed abuse patterns? - Does system degrade (fail-open) under sustained load?

Human Approval: - What's your approval latency distribution? - Does approval rate correlate with queue depth? - What context do approvers have? - At what queue depth does approval quality degrade?

6.3 Temporal Attack Surface Map

Create a **temporal attack surface map** for critical assets:

1. **Identify T_commit** for each attacker objective

2. **Map detection, decision, response times** for each control
3. **Identify race windows** where T_commit < T_respond
4. **Prioritize remediation** based on:
    - Likelihood (how easy to exploit?)
    - Impact (what's the objective value?)
    - Cost (how expensive to fix?)

---

7. Mitigation Strategies

7.1 Reduce Δt (Tighten Coupling)

Reduce W_j (faster detection): - Real-time streaming analytics instead of batch - Continuous monitoring instead of periodic scans - Inline enforcement instead of asynchronous

Reduce T_decide (faster decisions): - Automate where possible - Pre-compute decisions (offline analysis) - Simplify policy (reduce evaluation complexity)

Reduce A_j (faster response): - Automated response instead of manual - Pre-positioned countermeasures - Fail-fast instead of retry loops

Trade-off: Tighter coupling increases cost, reduces flexibility

7.2 Move T_commit Later (Delay Irreversibility)

Defer commitment: - Sign artifacts after all gates, not before - Grant minimal privileges initially, escalate on demand - Use short-lived credentials instead of long-lived - Enable quick revocation mechanisms

Trade-off: Increased complexity, potential usability impact

7.3 Adjust C_j (Fail-Closed Where It Matters)

Fail-closed for critical paths: - Break build on security gate timeout - Deny access on uncertain risk scores - Hard rate limit instead of degradation - Account lockout instead of CAPTCHA

Fail-open for non-critical: - Advisory findings for low-risk changes - Degrade service instead of deny for read-only operations - Alert instead of block for low-confidence detections

Trade-off: False positives, availability impact, operational overhead

7.4 Increase κ_j Tolerance (Accept Disruption)

Make security controls less disruptive: - Improve signal quality (reduce false positives) - Provide better context (help humans decide faster) - Reduce operational overhead (automate response) - Align incentives (don't punish security for usability)

Trade-off: Requires organizational change, not just technical

7.5 Remove Race Windows

Eliminate asynchrony where possible: - Inline security checks (no temporal gap) - Synchronous approval workflows (block until decided) - Real-time enforcement (detection ⟶ response with no delay)

Trade-off: Performance, scalability, cost

---

8. Broader Implications

8.1 "Zero Trust" as Temporal Minimization

"Zero trust" architectures implicitly minimize temporal attack surface by making commitment **continuous rather than binary**:

Traditional trust: Grant session → trust until revocation - **T_commit:** Session granted (long-lived) - **W_j:** Time until suspicious behavior detected - **Race window:** Attacker has entire session lifetime

Zero trust: Verify continuously → never fully committed - **T_commit:** Attempted for each action (minimized) - **W_j:** Real-time verification per request - **Race window:** Reduced to single action

Core principles map to temporal minimization: - **Continuous verification:** Reduce W_j to near-zero - **Least privilege:** Reduce T_commit impact (minimal access granted) - **Short-lived credentials:** Reduce exploitation window - **Inline policy enforcement:** Eliminate detection-response gap (T_respond → 0) - **Micro-segmentation:** Reduce lateral movement T_commit

Zero trust isn't about "never trust"—it's about **making T_commit small and frequent** rather than large and rare, so that W_j + A_j can keep pace.

Important caveat: Many "zero trust" deployments remain largely asynchronous (telemetry-driven policy enforcement), so the temporal attack surface can persist unless enforcement is truly inline and synchronous.

8.2 Human Latency as Fundamental Limit

Human-in-the-loop is the **slowest layer** in any security system: - Detection: Humans can observe (eventually) - Decision: Humans are slow and inconsistent - Response: Humans add hours-to-days latency

Implication: Any security control that requires human judgment is vulnerable to timing attacks.

Mitigation: Automate where possible, or accept the temporal attack surface.

8.3 AI/ML Security

The temporal attack surface framework applies directly to AI/ML security:

Model Evasion: - **T_commit:** Inference request completes - **W_j:** Adversarial detection scan time - **Race window:** If detection is slower than inference, evasion succeeds before detection

Data Poisoning: - **T_commit:** Model retraining with poisoned data - **W_j:** Anomaly detection in training pipeline - **Race window:** If poisoning affects model before detection, downstream attacks enabled

Prompt Injection: - **T_commit:** LLM generates output - **W_j:** Content filter processing time - **Race window:** If generation faster than filtering, harmful content released

Key insight: AI systems typically optimize for inference speed (low latency), creating inherent race windows against security controls that add overhead.

8.4 Observability ≠ Security

Having logs/telemetry (observability) is **necessary but not sufficient** for security.

The gap: Between when logs are generated and when they inform enforcement.

Example: Perfect logging of all actions is useless if: - T_detect (log processing) > T_commit (attacker objective) - Response is manual and takes days

Observability becomes security only when **T_detect + A_j < T_commit**.

8.4 Temporal Attack Surface > Traditional Attack Surface

Traditional "attack surface" focuses on: - Exposed services - Vulnerabilities - Access points

Temporal attack surface adds: - **Race windows - Asynchronous observers - Commitment points**

A system with no vulnerabilities still has temporal attack surface if: - Detection is slow - Decisions are asynchronous - Response is delayed

---

9. Discussion

9.1 Why This Framework Matters

Unifies disparate domains: Shows SIEM, CI/CD, auth, rate limiting, and human approvals share the same failure structure.

Provides actionable metrics: W_j, T_commit, C_j, κ_j are measurable in deployed systems.

Explains mitigation trade-offs: Precision vs. robustness, security vs. usability, automation vs. human judgment.

Generalizes beyond security: The Δt framework applies to any system where fast layers outrun slow layers (institutions, LLMs, distributed consensus).

9.2 Limitations

Simplified model: Real systems have: - Multiple detection layers with different W_j - Cascading decisions with complex dependencies - Political/organizational constraints beyond κ_j - Adaptive adversaries who learn defender parameters

No empirical validation: We provide framework and audit methodology, not measurements of production systems. **Future work should include:** (1) applying audit framework to real CI/CD pipelines, SIEM deployments, or authentication systems; (2) measuring W_j distributions in deployed controls; (3) demonstrating race windows with concrete T_commit measurements. Even a single measured case study would significantly strengthen the practical applicability.

Defender-focused: We deliberately avoid operational attack details, focusing on defender instrumentation and mitigation.

9.3 Future Work

Empirical measurement: Apply audit framework to real systems, measure W_j distributions, map temporal attack surfaces.

Adaptive models: Extend to handle defenders who adapt W_j, C_j based on observed attacks.

Multi-stage cascades: Model security as series of asynchronous observers, analyze compound temporal gaps.

Game-theoretic analysis: Formalize attacker-defender dynamics as differential game over temporal parameters.

Cross-domain applications: Apply temporal attack surface to non-security domains (safety-critical systems, financial controls, healthcare).

---

10. Conclusion

We have shown that asynchronous security systems—SIEM, CI/CD, authentication, rate limiting, human approval—share a universal vulnerability structure: **the temporal gap between detection and response**.

Building on the Δt framework and its application to DPI circumvention, we formalized the **Temporal Attack Surface** as the set of race windows where attacker commitment outruns defender response. The Universal Bypass Inequality

provides conditions under which these systems fail: **when evidence arrives too late, costs too much to process, or doesn't exist—and enforcement happens after commitment**.

We demonstrated this structure across five security domains and provided defender instrumentation to measure temporal vulnerabilities. Key insights:

1. **All asynchronous security is a race** between attacker commitment and defender response
2. **Humans are the slowest layer** and therefore most vulnerable to timing attacks
3. **Fail-open defaults** are common (optimizing for availability) and exploitable
4. **Political budget (κ_j)** constrains security controls as much as technical capability
5. **"Zero trust" implicitly minimizes temporal attack surface** by reducing detection-response gaps

The precision-robustness trade-off explains why defenders move toward crude enforcement as attacks mature: precision requires tight temporal coupling (vulnerable to Δt attacks), robustness requires loose coupling (high false positive rate).

The unifying insight: Security failures in asynchronous systems are **control-theoretic race conditions**, not configuration errors. Defense requires understanding temporal dynamics, not just access control or vulnerability management.

The Temporal Attack Surface is not a new problem—it's a **lens** that reveals existing vulnerabilities through their temporal structure. Systems that appear secure under traditional analysis fail when attackers are simply **faster** than defenders.

References

[1] Beck, J. "Temporal Asymmetry in Censorship Systems." Zenodo, 2026. DOI: 10.5281/zenodo.18235697

[2] Beck, J. "The Coherence Criterion: A Unified Framework for Stability in Hierarchical Systems." Zenodo, 2025. DOI: 10.5281/zenodo.17726789

[3] Beck, J. "Detecting Temporal Debt in Language Models and Software Systems." Zenodo, 2025. DOI: 10.5281/zenodo.17859324

[4] Beck, J. "The Second Law of Organizations: How Temporal Lag Drives Irreversible Institutional Entropy." Zenodo, 2025. DOI: 10.5281/zenodo.17726889

[5] Beck, J. "Capacity-Constrained Stability: A Control-Theoretic Framework for Institutional Resilience." Zenodo, 2025. DOI: 10.5281/zenodo.18019051

[6] Beck, J. "You Need More Than Just Attention: Invariant Requirements for Temporal Coherence in AI Systems." Zenodo, 2025. DOI: 10.5281/zenodo.18039926

[7] Ashby, W.R. "An Introduction to Cybernetics." Chapman & Hall, 1956.

[8] NIST. "Zero Trust Architecture." NIST Special Publication 800-207, August 2020.

[9] Saltzer, J. & Schroeder, M. "The Protection of Information in Computer Systems." Proceedings of the IEEE, 1975.

[10] Anderson, R. "Security Engineering: A Guide to Building Dependable Distributed Systems." Wiley, 2020.

[11] Schneier, B. "Attack Trees." Dr. Dobb's Journal, December 1999.

[12] Hutchins, E., Cloppert, M., & Amin, R. "Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains." Lockheed Martin, 2011.

[13] MITRE Corporation. "MITRE ATT&CK: Adversarial Tactics, Techniques, and Common Knowledge." https://attack.mitre.org/ (accessed January 2025).

[14] Bishop, M. & Dilger, M. "Checking for Race Conditions in File Accesses." Computing Systems, Vol. 9, No. 2, 1996.

[15] Vigna, G. & Kemmerer, R. "NetSTAT: A Network-based Intrusion Detection System." Journal of Computer Security, Vol. 7, No. 1, 1999.

Appendix A: Defender Audit Checklist

For each security control in your system:

1. Temporal Parameters - [ ] Measured W_j distribution (median, p95, p99) - [ ] Identified T_commit for critical objectives - [ ] Documented C_j (fail-open or fail-closed) - [ ] Estimated κ_j (false positive tolerance) - [ ] Measured A_j (response latency)

2. Race Window Analysis - [ ] T_commit < T_detect? (Race window exists) - [ ] T_commit < T_decide? (Decision too slow) - [ ] T_commit < T_respond? (Response too late) - [ ] Can attackers complete objectives in one W_j window?

3. Commitment Analysis - [ ] Where is point of no return? - [ ] Can actions be reverted? - [ ] What's the revert cost/latency? - [ ] Are decisions reversible without detection?

4. Fail-Open Defaults - [ ] Which controls skip on timeout? - [ ] Which findings are advisory-only? - [ ] What happens under load? - [ ] What happens under uncertainty?

5. Human Bottlenecks - [ ] Which controls require human approval? - [ ] What's approval latency distribution? - [ ] Does approval rate correlate with queue depth? - [ ] At what queue depth do approvals become rubber-stamps?

6. Mitigation Prioritization - [ ] Which race windows are exploitable? - [ ] What's the impact of each? - [ ] What's the cost to fix? - [ ] What's the κ_j constraint?

Acknowledgments: This work emerged from collaborative formalization across multiple AI systems (ChatGPT, Gemini, Claude), demonstrating multi-model semantic amplification for theoretical development. The generalization from DPI circumvention to asynchronous security was identified through cross-domain pattern recognition. All errors remain the author's responsibility.

Ethical Statement: This research is presented for defender instrumentation and threat modeling. We deliberately avoid operational exploitation details and frame all analysis from the defender perspective. The goal is to help organizations identify and mitigate temporal attack surface, not to enable attacks.

Conflict of Interest: The author has no financial interests in security products or services.