itle**Detecting Reseller Bots in Limited Clothing Drops Using Risk Scoring and Navigation Patterns**

itle**Detecting Reseller Bots in Limited Clothing Drops Using Risk Scoring and Navigation Patterns**

Godfred Tekpor

February 5, 2026

1 Project Topic: Detecting Reseller Bots in Limited Clothing Drops Using Risk Scoring and Navigation Patterns

1.1 Motivation and Significance

Limited-release clothing drops create extreme bursts of traffic and strong incentives for automation. Reseller bots attempt to gain an advantage by executing faster-than-human navigation paths (product → add-to-cart → checkout), polling inventory endpoints, and distributing attempts across many accounts or proxies. In practice, purely rule-based defenses (e.g., hard blocks and simple rate limits) are often either too weak against advanced bots or too costly for legitimate customers. This project focuses on a practical middle ground: a risk scoring system that estimates bot-likelihood from session navigation patterns and request timing, then applies calibrated mitigation only when needed.

Prior work shows that navigation and timing signals in web logs can be used to detect evasive bots, and that combining multiple evidence sources can improve robustness. Iliou et al. demonstrate that modeling sessions from web logs can be effective, and that adding human-interaction evidence (mouse behavior) raises attacker cost and improves detection under an "advanced bot" threat model [**?**]. However, bot detection is not a static problem. Iliou et al. also show an adversarial dynamic where reinforcement-learning bots learn navigation behaviors that evade detectors trained on web-log features, and can re-adapt after the detector is retrained [**?**]. This motivates designing a risk score that can be stress-tested against adaptive policies rather than only simple scripted bots. Finally, risk scoring is only valuable if it connects to mitigation decisions. Gangwal et al. propose a multi-barrier CAPTCHA mechanism that balances usability and security, suggesting a pathway for escalation when a session appears suspicious [**?**]. Together, these sources motivate a layered approach: (1) score sessions using navigation patterns, (2) escalate friction for high-risk traffic, and (3) evaluate robustness under evasion.

1.2 Goals and Contributions

The project will produce a working software prototype ("DropGuard") and an experimental evaluation. The main goals are:

1. Build a small drop-style e-commerce site (product, cart, checkout) that records detailed session-level event logs.

2. Implement multiple bot simulators representing distinct reseller behaviors (fast-path, burst/polling, stealth) and a human-session simulator.

3. Design an interpretable risk scoring engine using navigation-path and request-timing features derived from web logs.

4. Evaluate tradeoffs between security and user friction by comparing mitigation policies (allow, delay, throttle, challenge).

5. Stress-test robustness by introducing adaptive behavior inspired by reinforcement-learning evasion results [**?**].

The intended contribution is not to claim a universal defense against all automation, but to provide a measurable, reproducible study: which navigation-pattern features provide the most signal, how early scoring decisions can be made, and how mitigation choices shift outcomes (bot capture rate vs. legitimate user friction).

1.3 Software Description

DropGuard will be implemented in Python using FastAPI and a SQLite event store. The system will treat each visitor as a session with a unique session identifier. Each request (page view or API call) will be logged as an event with timestamp, endpoint, response status, and minimal metadata needed for analysis. A risk engine will periodically compute a session score in $[0, 1]$ based on features such as:

- **Speed features:** time from product-view to add-to-cart, add-to-cart to checkout, median inter-event gap.
- **Path features:** step skipping, repeated loops, unusually short navigation sequences.
- **Burstiness:** request rate spikes, polling frequency, retry patterns.

Mitigation will be applied at sensitive actions (add-to-cart and checkout). Low-risk sessions proceed normally; medium-risk sessions may be delayed or rate-limited; high-risk sessions may be challenged (e.g., a stronger interaction step) or throttled. This approach aligns with research showing that combined evidence and escalation can improve effectiveness while keeping the user experience reasonable [**?**, **?**].

1.4 Evaluation Plan

Experiments will simulate repeated "drops" with limited inventory and concurrent human and bot sessions. Metrics will include:

- **Bot capture rate:** percentage of inventory purchased by bots.
- **Human success rate:** percentage of legitimate sessions completing checkout.
- **False positives:** legitimate sessions incorrectly challenged or throttled.

- **Friction cost:** added time to checkout and drop-off rate due to mitigation.
- **Robustness:** performance under stealth bots and adaptive behavior inspired by RL evasion [**?**].

1.5 Logic Diagram

Figure **??** shows the high-level flow of the system. The user interacts with the drop website, events are logged, a risk score is computed, and policy decides whether to allow, add friction, or throttle.

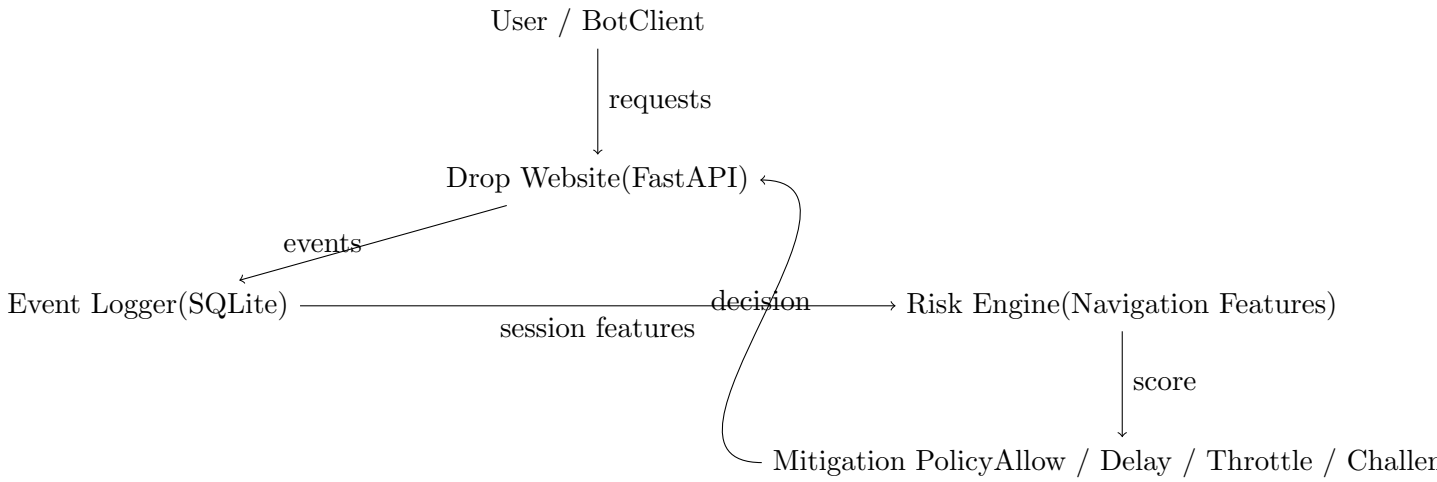

Figure 1: DropGuard logic flow: navigation events → risk scoring → mitigation at sensitive actions.

Placeholder for UI sketch 1 (e.g., Product + Cart + Checkout screens).

Figure 2: UI sketch placeholder 1 (replace with an image file in your repo).

Placeholder for UI sketch 2 (e.g., Risk score dashboard / admin view).

Figure 3: UI sketch placeholder 2 (replace with an image file in your repo).

A Feature List (Development Roadmap)

1. Implement product page endpoint and static product inventory model.
2. Implement cart endpoints (add, remove, update quantity/size).
3. Implement checkout endpoint and simulated "purchase" action that decrements inventory.
4. Create session management (session ID cookies) for all clients.
5. Implement structured event logging to SQLite (timestamp, session ID, endpoint, status, metadata).
6. Implement feature extraction for navigation patterns from event logs (timing, path, burstiness).
7. Implement risk scoring function returning a bot-likelihood score in $[0, 1]$.
8. Implement mitigation policy gates at add-to-cart and checkout based on risk score.
9. Build bot simulator scripts: fast-path bot, burst/polling bot, stealth bot.
10. Build a human-session simulator with realistic timing variance and navigation noise.
11. Implement experiment runner to launch mixed populations and run repeated drop trials.
12. Implement evaluation metrics and produce summary tables/plots (capture rate, false positives, friction).
13. **Stretch:** Add an adaptive bot that tunes timing/path choices to reduce risk score (stress test).
14. **Stretch:** Add a simple admin dashboard to visualize sessions, scores, and mitigation decisions.

References