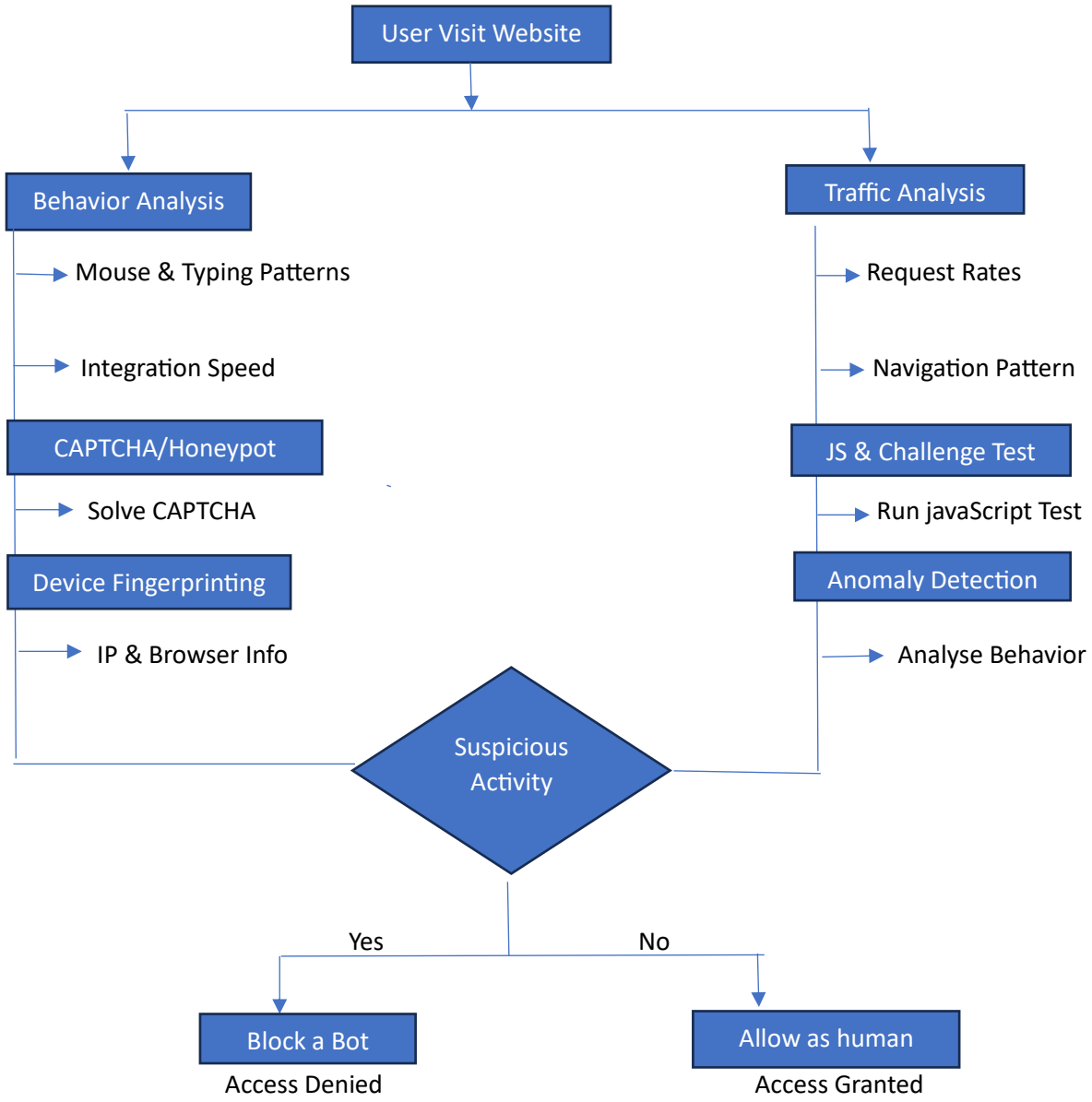
# Task : Detecting Automated Traffic

Submitted to: Web3Task Pvt. Ltd

Automated traffic from bots can impact website security, performance, and data accuracy. While some bots are useful, malicious one may scrape data, overload systems, or attempt unauthorized access. Identifying and managing bot activity is essential for protecting modern web applications. This document highlights key indicators of bot behavior, detection techniques, and mitigation strategies.




---

## 1. Indicators of Bot Activity

Bot exhibit distinct behavioral and technical patterns from human interaction:

- Unnatural Mouse Movements :- Straight-line or perfectly uniform mouse paths. Humans typically move with small deviations and irregular patterns.
- Unusual Request Frequency :- Bots often generate abnormally high request volumes (e.g:- 100+ requests per second from a single IP ), while humans usually interact at a slower pace (1-5 requests per minute).

Example:- a sudden spike in API calls to a non-public end points from a new IP with no prior activity suggests bot activity.

- Instant Form Filling:- Forms completed too quickly or perfectly (e.g:- all fields filled within milliseconds).
- Input Behavioral Anomalies :- Bots submit forms too quickly (e.g:- 0.1 sec delay between page load and submission) or use repetitive credential combinations. Humans typically pause or correct inputs.
- Abnormal Click Pattern:- Clicking elements in the exact same position repeatedly or at extremely high speed.
- Rapid Page Navigation :-Moving through multiple pages in an implausibly short time.
- Session Duration and Navigation patterns:-Very short sessions (e.g under 2 sec ) with rigid, sequential navigation patterns typical of automated crawling.
- Technical Artifacts:- Bots often use suspicious user agents (e.g:- "Bot/1.0" instead of standard browser), lack required HTTP headers (e.g:- no Accept-language ) or show geolocation mismatch (an IP from Germany with a browser language set to Japanese).

## 2. Detection Methods

- Device and Browser Fingerprinting:- Bots often have generic or inconsistent fingerprints. Headless browsers may lack plugins , fonts ,or WebGL properties, or navigator properties can reveal automation.
- Honeypot trap:- Detect bots by placing elements real human users won't interact with but automated bots likely will For example:- A hidden form field, a fake link that users never see .
- Machine Learning Anomaly Detection:- Train models on historical traffic data to identify deviations such as sudden spikes in 404 error . Algorithms like Isolation Forests or LSTM networks detect subtle patterns missed by rule-based systems.
- Rate Limiting and Adaptive Thresholds:- Apply dynamic requests limits per IP address(e.g:- 50 requests per minute) because bots have ability to avoid disrupting legitimate activity. Tools like Nginx or Cloudflare enable this at the infrastructure level.


## 3. Prevention & Mitigation

Mitigation must prioritize user experience while blocking bots.

- Progressive Challenges:- Escalate verification only when bot-like is detected. Start with rate limiting then trigger CAPTCHA . Avoid blanket CAPTCHA to prevent frustration.
- Bot Whitelisting/ Blacklisting:- Allow trusted bots e.g:- Googlebot is a reverse DNS checks while blocking IP from threat intelligence feeds e.g:- AllienVault OTX. Use WAF rules to block malicious traffic at the edge (e.g:- cloudeflare's Bot Management).
- Use Centric Optimization:- Recognize returning users via cookies to bypass challenges after initial verification. For example , a returning user with valid sessions token should skip CAPTCHA.

## 4. Engineering Approach and Decision Rationale

- From an engineering perspective, detection should be scalable, efficient, and user-centric. I would implement a layered pipeline where inexpensive checks such as request rate monitoring and header validation are performed first to handle large traffic volumes with minimal overhead. Sessions that trigger suspicion would then undergo behavioral analysis which provides stronger signals but requires additional processing.

  Rather than immediately blocking traffic, progressive challenges would reduce false positive and maintain usability. Only repeated violations would lead to stricter enforcement such as rate limiting or blocking. This staged decision model balances infrastructure cost, detection accuracy, and user experience-key considerations when deploying solutions in real production environments.