International Journal of Engineering Research & Science (IJOER) ISSN: [2395-6992] [Vol-12, Issue-3, March 2026]

Using Honeypots to Identify Bot-Driven Web Attacks

Kshiteeja Churi[1*]; Garrgi Phadke[2]; Purnima Singh[3]

Department of Master of Computer Applications, University of Mumbai, India
*Corresponding Author

Abstract— *Web applications are increasingly targeted by automated bot-driven attacks such as credential stuffing, spam submissions, brute-force login attempts, and form abuse. Traditional defence mechanisms like CAPTCHA often degrade user experience while still being bypassed by advanced bots. Honeypot-based security mechanisms provide a lightweight and user-transparent alternative for detecting malicious automated behaviour. This paper presents a honeypot-based approach to identify bot-driven web attacks by embedding deceptive elements such as hidden form fields and fake authentication endpoints within web applications. Legitimate users do not interact with these elements, whereas bots often trigger them during automated crawling or submission processes. The proposed system logs and analyses honeypot interactions to classify malicious traffic effectively. Experimental evaluation using simulated bot traffic demonstrates that honeypot techniques can achieve high detection accuracy with minimal impact on genuine users. This approach offers a cost-effective, scalable, and user-friendly solution for enhancing web security.*

Keywords— Automated Attacks, Bot Detection, CAPTCHA Alternative, Honeypot, Web Security.

I. INTRODUCTION

Web applications play a vital role in modern digital services, supporting activities such as online communication, e-commerce, data sharing, and service delivery. Due to their widespread usage and accessibility, web applications have become frequent targets of automated bot-driven attacks. These attacks include spam form submissions, credential stuffing, brute-force login attempts, web scraping, and denial-of-service activities. Such malicious automation can compromise sensitive data, overload resources, and degrade overall system performance, posing serious security challenges for organizations.

Traditional security mechanisms such as CAPTCHA are commonly implemented to differentiate between human users and automated bots. Although CAPTCHAs provide a basic level of protection, they often negatively impact user experience by increasing interaction complexity and reducing form completion rates. Additionally, modern bots equipped with machine learning techniques and third-party CAPTCHA-solving services have significantly reduced the effectiveness of CAPTCHA-based defences. This has created a demand for alternative security approaches capable of detecting automated threats without disrupting legitimate users.

Honeypot-based security techniques offer a practical and user-transparent solution. A honeypot is a deceptive security mechanism designed to attract and identify malicious behaviour by presenting resources or interaction points that legitimate users do not access. In web security, honeypots can be implemented using hidden form fields, invisible input elements, or fake login endpoints. Human users do not interact with these elements, whereas automated bots frequently trigger them during form submissions or crawling activities, thereby revealing their presence.

This research focuses on using honeypot techniques to identify bot-driven web attacks in web applications. By embedding low-interaction honeypots within web forms and authentication systems, the proposed approach enables effective detection of automated attacks with minimal computational overhead and no impact on user experience. The system records and analyses honeypot interactions to classify malicious traffic accurately. The solution is simple, cost-effective, and suitable for small to medium-scale web applications, making it an efficient alternative to traditional bot detection mechanisms.

II. RELATED WORK

Several studies have investigated honeypot and deception techniques for cybersecurity and bot/attack detection. These works provide context and foundational insights for the proposed honeypot-based detection system.

Ahmad et al. [1] proposed a system for detecting and analysing active cyber attacks using honeypots. Their research focuses on how honeypots can capture attack activities and assist in understanding the nature and source of malicious behaviour. While effective in capturing real attacks, their work does not specifically address web form bot detection.

Abewa and Melese [2] developed a dynamic interactive honeypot for web application security that can detect multiple types of web intrusion attempts including SQL injection, cross-site scripting, and path traversal. Their approach integrates scalable interaction levels and robust logging to capture attacker behaviour. Although it enhances detection of various attacks, additional customization is required for specific web attack types.

Wang et al. [3] investigated honeypot detection techniques in advanced botnet attacks, focusing on how attackers recognize and evade honeypots. This study provides insights into botnet behaviour and honeypot vulnerabilities, highlighting the need for resilient honeypot designs. However, it primarily addresses structural botnet detection rather than web form or credential-based attacks.

Priya and Chakkaravarthy [4] explored containerized cloud-based honeypot deception techniques to track attackers. Their framework demonstrates deployment of honeypot environments in cloud infrastructure to analyse attacker interactions and identify malicious behaviour. Despite strengths at the infrastructure level, it does not focus on form-based web applications.

McKenna [5] investigated web robot detection using honeypots by embedding hidden resources, such as forms and URLs invisible to humans but accessible to automated bots. The study showed that monitoring interactions with these trap resources effectively differentiates human and automated activity, providing insights into bot behaviour and improving detection accuracy. This concept is directly relevant to detecting automated web form attacks in modern honeypot systems.

III. TECHNOLOGIES USED

This section describes the key technologies employed in the proposed honeypot-based bot detection system. The study combines web security mechanisms, client-side scripting, and server-side processing to identify bot-driven web attacks. The system is designed using standard web technologies and the Django framework to ensure simplicity, scalability, and compatibility with modern web applications.

3.1 HTML/CSS

The frontend interface is built with HTML and CSS. Forms are created for user input, while honeypot fields are hidden using CSS to remain invisible to humans but detectable by bots. Although invisible to human users, these hidden honeypot fields remain fully accessible within the HTML source code, allowing automated bots that parse and auto-fill form elements to interact with them. Many bot scripts are designed to complete all available input fields without distinguishing between visible and hidden elements, which results in the honeypot field being filled unintentionally. This behaviour serves as a strong indicator of automated activity.

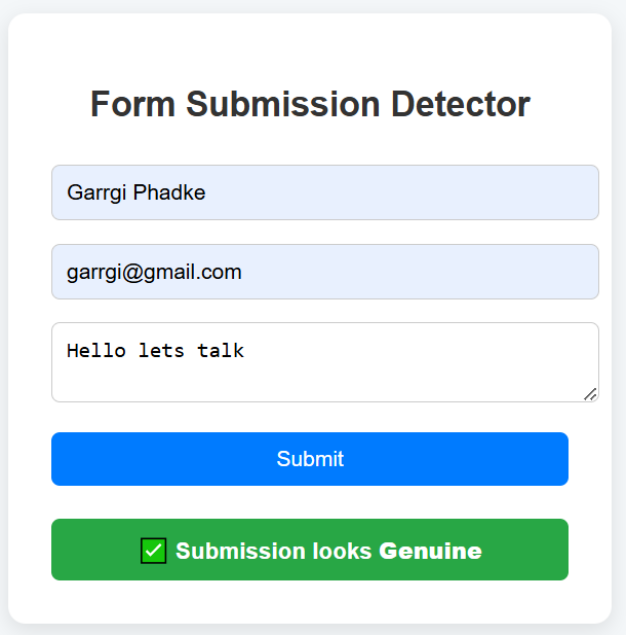

FIGURE 1: Highlight of honeypot field in the form

3.2 JavaScript

JavaScript enhances interactivity and performs client-side checks. The system inspects input for spam keywords and validates honeypot fields before sending requests to the server. This allows immediate feedback to the user and adds a secondary layer of detection for suspicious behaviour.

```
// Honeypot or spam check
if (honeypot.trim() !== "") {
  resultBox.innerHTML = "⚠️ Detected as <b>Fake Submission</b> (Honeypot triggered)";
  resultBox.className = "fake";
} else if (isSpam) {
  resultBox.innerHTML = "⚠️ Detected as <b>Fake Submission</b> (Spam content found)";
  resultBox.className = "fake";
} else {
  resultBox.innerHTML = "✅ Submission looks <b>Genuine</b>";
  resultBox.className = "genuine";
}
```

FIGURE 2: Honeypot check

3.3 Honeypot Field

Hidden honeypot fields act as decoys for bots. When a bot fills this field, the submission is flagged as malicious. The field is not visible to human users, reducing false positives. Any interaction with this field triggers logging for later analysis.

```
  <!-- Honeypot field -->
  <input type="text" id="website" class="honeypot" placeholder="Do no fill this">
  <button type="submit">Submit</button>
</form>
```

FIGURE 3: Sample honeypot log entries

3.4 Logging Module

The system implements logical logging through client-side detection outputs. Each form submission is classified in real time as fake or genuine based on honeypot activation and spam keyword detection. The classification result represents a logical log entry that can be extended to server-side storage in future implementations.

3.5 Django Web Framework

Django serves as the backbone for server-side processing. It handles request routing, form submission logging, and honeypot detection logic. When a bot interacts with the hidden honeypot field or submits abnormal requests, Django logs the event for further analysis. Its middleware and database integration make the system scalable and efficient for real-world deployment.

3.6 Rule-Based Detection Logic

The proposed detection mechanism employs a rule-based approach to identify bot-driven form submissions by evaluating each request against predefined conditions. Key rules include the activation of hidden honeypot fields (which normal users will not interact with) and the presence of suspicious or spam-related keywords in form inputs. If a submission violates any of these rules, it is flagged as bot-driven. This approach is computationally lightweight, enabling real-time detection without affecting normal user experience.

IV. PROPOSED SYSTEM

The proposed system is designed to silently detect bot-driven web attacks using honeypot techniques, without interrupting legitimate users. Unlike CAPTCHA-based mechanisms, the system relies on deceptive frontend elements and server-side request analysis to identify automated behaviour.

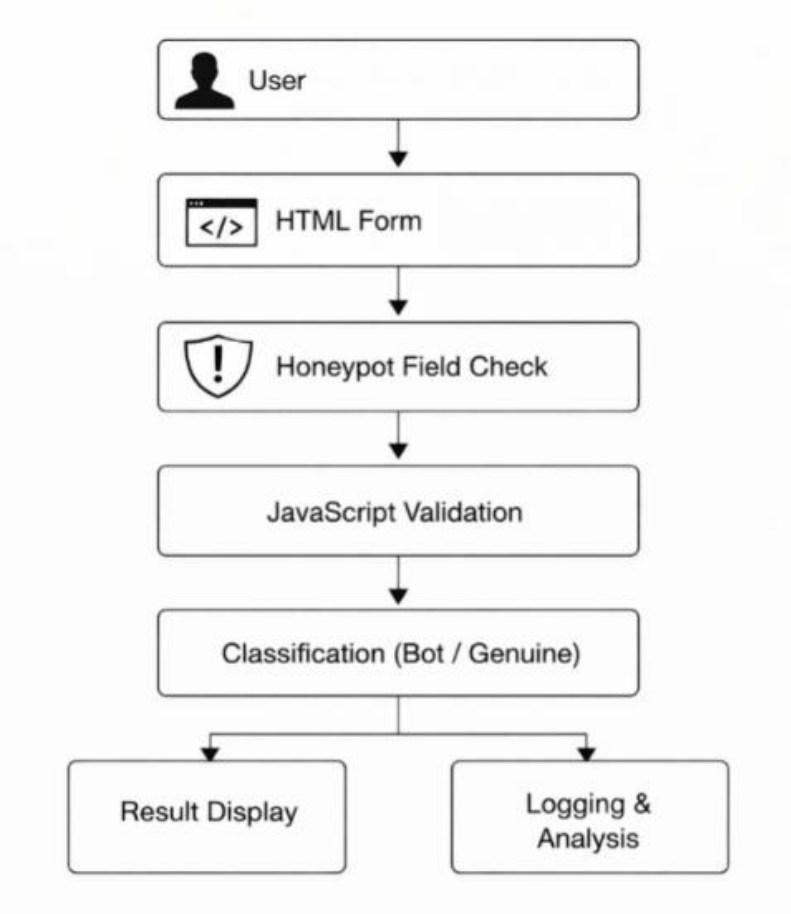

FIGURE 4: Block diagram of the proposed system

4.1 User Interface and Honeypot Injection

The system begins with a web-based form developed using HTML and CSS. Along with standard visible input fields such as name, email, and message, a hidden honeypot field is injected into the form. This honeypot field is concealed from human users using CSS properties such as display: none, but remains accessible to automated bots that parse HTML code directly.

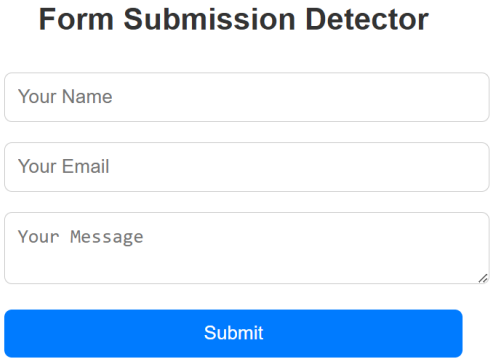

FIGURE 5: Form submission detector

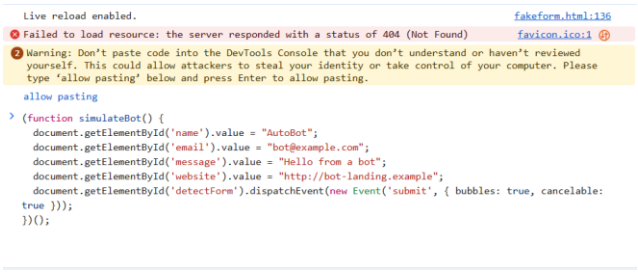

FIGURE 6: Honeypot auto entry code

4.2 Client-Side Request Monitoring

Once the user submits the form, JavaScript-based validation is triggered. This module performs lightweight checks to detect whether the honeypot field has been filled, scans the message content for common spam keywords, and prevents unnecessary server requests in case of obvious bot activity.

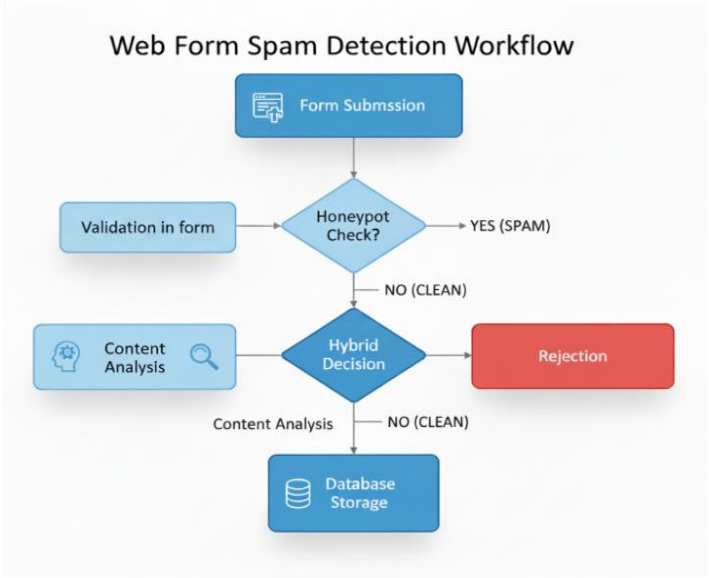

FIGURE 7: Web form spam detection workflow

4.3 Server-Side Processing and Logging

After form submission, all requests are forwarded to the Django backend, where server-side validation ensures data integrity and security. The backend specifically monitors interactions with the hidden honeypot field and other indicators of automated submissions. Any suspicious activity is recorded in structured logs, capturing essential details such as the user's IP address, timestamp of the request, User-Agent string, and the reason for detection.

4.4 Bot Classification and Decision Module

The Bot Classification and Decision Module analyzes each incoming form submission to determine whether it originates from a genuine user or an automated bot. This module employs a rule-based classification approach, in which predefined conditions are evaluated based on observed behaviour during form submission.

TABLE 1
CONDITION VS. DETECTION RESULT

Condition	Detection Result
Honeypot field filled	Bot detected
Spam keywords present	Suspicious
Normal form submission	Genuine user

4.5 Alert and Response Mechanism

Once a submission is classified as bot-driven, the system activates an appropriate alert and response mechanism. Depending on the severity and frequency of detected behaviour, the system may display a warning message, temporarily block repeated submissions, or enforce IP-based rate limiting to prevent automated attacks.

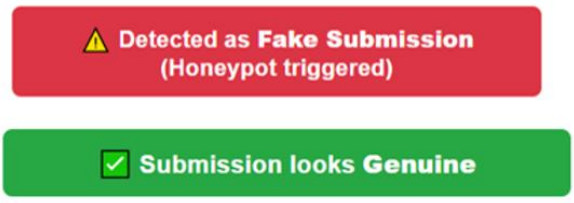

FIGURE 8: Submission detection

TABLE 2
DIFFERENCE BETWEEN CAPTCHA-BASED DETECTION AND HONEYPOT-BASED DETECTION

Parameter	CAPTCHA-Based Detection	Honeypot-Based Detection
User Interaction	Requires user action (text/image selection)	No user interaction required
Impact on User Experience	Can be intrusive and time-consuming	Completely transparent to users
Accessibility Issues	Difficult for visually impaired users	Fully accessible
Resistance to Advanced Bots	Can be bypassed by AI-based solvers	Effective against basic and automated bots
Implementation Complexity	Moderate to high	Low
Server Overhead	Higher due to validation services	Minimal
Privacy Concerns	Often relies on third-party services	No third-party dependency

V. RESULTS AND DISCUSSION

The proposed honeypot-based bot detection system was evaluated using a combination of genuine user interactions and automated form submission scripts designed to simulate bot behaviour. During testing, automated scripts consistently interacted with the hidden honeypot field, which served as a reliable indicator of non-human activity. Consequently, these submissions were accurately identified and classified as bot-driven attempts. In contrast, legitimate users completed only the visible form fields and did not trigger the honeypot, resulting in correct classification as genuine submissions.

Client-side validation contributed significantly to detecting suspicious content at an early stage by identifying repeated spam-related keywords and preventing unnecessary server requests. This early filtering reduced server load and enhanced overall system efficiency. On the server side, structured logging facilitated effective monitoring of suspicious requests by recording details such as IP address, timestamp, and reason for detection. These logs proved valuable for analysing attack patterns and understanding the behaviour of automated bots over time.

The experimental results demonstrate that the proposed system effectively detects automated attacks while maintaining a seamless experience for legitimate users. Unlike CAPTCHA-based mechanisms, which often disrupt user interaction and introduce accessibility challenges, the honeypot-based approach operates transparently in the background. Furthermore, the absence of third-party dependencies minimizes privacy concerns and computational overhead.

VI. CONCLUSION

This research presents a simple yet effective honeypot-based approach for identifying bot-driven web attacks in form-based web applications. By integrating hidden honeypot fields within the user interface, implementing client-side validation for early detection, and performing server-side analysis and logging using the Django framework, the proposed system successfully distinguishes between genuine users and automated bots without interrupting normal user activity.

Unlike traditional CAPTCHA mechanisms, which require explicit user interaction and may negatively impact usability and accessibility, the proposed solution operates silently and transparently. This enhances user experience while maintaining adequate security against automated submissions. The lightweight nature of the system, combined with its ease of implementation using standard web technologies, makes it suitable for deployment in real-world web environments.

Future enhancements to this work may include:

- Adaptive rule refinement based on logged bot behaviour

- Integration of rate-limiting strategies
- Combination of honeypot techniques with machine learning-based analysis for advanced threat detection

Overall, the proposed honeypot-based detection mechanism offers a scalable, efficient, and user-friendly solution for mitigating bot-driven web attacks.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

REFERENCES

[1] McKenna, S. (2008). Using honeypots for web robot detection. In *Proceedings of the International Conference on Web Intelligence*.

[2] Ahmad, W., & [Full author names if available]. (2023). Detection and analysis of active cyber attacks using honeypots. *International Journal of Computer Networks and Security*.

[3] Abewa, I. Y. T., & Melese, S. Z. (2024). Dynamic interactive honeypot for web application security. *Journal of Information Security and Applications*.

[4] Wang, P., & [Full author names if available]. (2010). Honeypot detection in advanced botnet attacks. *International Journal of Network Security, 11*(3).

[5] Priya, V. S. D., & Chakkaravarthy, S. S. (2023). Cloud-based honeypot deception framework for cyber attack detection. *Journal of Cloud Computing*.

[6] Kreibich, C., & [Full author names if available]. (2005). Honeycomb: Creating intrusion detection signatures using honeypots. *ACM SIGCOMM Computer Communication Review*.

[7] Bursztein, E., & [Full author names if available]. (2014). The failure of CAPTCHA and alternatives for bot detection. In *IEEE Symposium on Security and Privacy*.

[8] Stone-Gross, B., & [Full author names if available]. (2011). Your botnet is my botnet: Analysis of web-based malware. In *Proceedings of the ACM Conference on Computer and Communications Security*.

[9] Nazir, S., & [Full author names if available]. (2017). Low-interaction honeypots for web-based attack detection. *International Journal of Advanced Computer Science and Applications*.

[10] Almotairi, S., & [Full author names if available]. (n.d.). Survey of bot detection techniques for web applications. *Journal of Network and Computer Applications*.