CHAPTER ONE

INTRODUCTION

1.1 Background of the Study

In today's digital world, web applications are everywhere—powering services we depend on daily, from online banking and shopping to healthcare and education. As these applications become more advanced and interconnected, they also become more exposed to cyber threats. Attackers often target common features such as login pages, search bars, file upload tools, and API endpoints—elements that are essential for functionality but are sometimes overlooked during routine security checks.

These components are frequent entry points for attacks like brute-force logins, SQL injection (SQLi), cross-site scripting (XSS), and malicious file uploads (Chen et al., 2023). Traditional cybersecurity tools—like firewalls and antivirus software—are mostly reactive. They block threats when they're detected, but they don't always offer insight into how or why the attack happened.

To stay ahead of increasingly sophisticated attackers, security experts are turning to more proactive strategies. One such method is the use of honeypots—deceptive elements embedded within web applications that mimic real, vulnerable systems. These traps are designed to attract and engage malicious users, allowing organizations to monitor their activity in a controlled environment (Rahman & Malik, 2024).

Unlike traditional security tools, honeypots don't just block attacks—they observe them. By acting as convincing decoys (like fake login forms, mock API routes, or dummy file upload pages), honeypots lure attackers into interacting with them. While attackers believe they've found a real target, the honeypot quietly records their actions, tools, and techniques in real time. This gives security teams valuable intelligence on how threats evolve and where defenses need to improve (Kumar & Zhang, 2022).

The integration of honeypots into live web applications marks a major shift in how organizations approach cybersecurity. Rather than waiting for a breach to occur, these systems provide early

warnings by detecting suspicious behavior before real damage is done. They also assist with digital forensics, helping teams understand the source and intent of attacks.

What makes web-based honeypots especially powerful is their dual role: protection and education. They don't just stop threats—they teach us about them. With each attempted attack, a honeypot captures patterns and behaviors that help teams refine their security strategies. In an era where threats are often automated and fast-moving, this kind of adaptive defense is crucial (Ibrahim et al., 2025).

Ultimately, the use of honeypots within web applications reflects a smarter, more strategic approach to cybersecurity. Instead of just building higher walls, organizations are setting traps—not to retaliate, but to learn. In doing so, they're turning their web applications into intelligent systems capable of anticipating risks and responding to them before harm is done.

1.2 Statement of the problem

Despite advancements in cybersecurity, many web applications remain vulnerable due to limitations in traditional defense mechanisms. The following key problems highlight the need for more proactive and intelligent security measures:

1. **Traditional defenses are reactive and often too late** Firewalls, intrusion detection systems (IDS), and other conventional tools are designed to block or report threats after they occur. However, many targeted attacks—especially those exploiting login panels, API routes, or user input fields—are not detected until significant damage has already been done (Nguyen & Okafor, 2023).
2. **Common web components are frequently exploited** Features like administrative login forms, poorly secured APIs, and unsanitized input fields are regularly targeted by automated scripts and bots. These entry points are often left exposed or weakly protected, making them attractive to attackers seeking easy access.
3. **Lack of proactive deception strategies in most web applications** While traditional preventive tools are widespread, the use of deception-based security—like honeypots—is still limited in mainstream web applications. Many organizations focus solely on blocking threats rather than studying them (Patel & Singh, 2024).

4. **Missed opportunities to learn from real-world attacks** Without systems that monitor and analyze attacker behavior, organizations fail to gather valuable insights from intrusion attempts. This prevents them from adapting their defenses and staying ahead of evolving attack methods.
5. **Existing honeypot systems often lack realism** Many honeypots are too simplistic or isolated from real application environments. As a result, experienced attackers can easily detect and avoid them, reducing their value for data collection and threat analysis (Owolabi et al., 2022).
6. **Need for integrated, intelligent deception in live applications** There is a clear need for honeypot systems that are embedded directly into the structure of a web application—disguised as realistic features such as fake login forms, decoy APIs, or dummy file uploads. These traps should include smart logging capabilities to safely collect attacker behavior data without disrupting real user experiences.

1.3 Objectives of the Study

The study is guided by the following specific objectives:

1. To investigate how deceptive web elements—such as login forms, search boxes, API endpoints, and file upload components—can be designed to mimic legitimate application features convincingly.
2. To develop a system for monitoring and logging interaction attempts with these decoy components, capturing key metadata such as IP addresses, timestamps, user-agent strings, input payloads, and potentially malicious patterns.
3. To evaluate the role of simulated system responses (e.g., “Login failed,” “Access denied,” “Database error”) in maintaining attacker engagement and realism.
4. To analyze the behavior of attackers interacting with the honeypot system, identifying trends such as brute-force login attempts, injection techniques, and malicious file types.
5. To compare the effectiveness of integrated honeypot mechanisms in detecting unauthorized access attempts versus traditional passive defense tools.
6. To propose strategies for integrating honeypot-based deception systems into production web applications for improved threat detection and security intelligence.

1.4 Scope of the Study

This study is centered on the design, development, and deployment of a client-server honeypot system embedded within a simulated web application environment. The scope is confined to specific components and clearly defined boundaries, as outlined below:

1. **Deceptive Web Interfaces** The system includes four primary honeypot elements designed to mimic typical web application features and attract malicious activity:

    - A **fake login page** to detect brute-force authentication attempts.
    - Simulated **API endpoints** intended to lure bots and automated scanners.
    - A **search form** aimed at exposing SQL injection (SQLi) and cross-site scripting (XSS) attack patterns.
    - A **file upload module** to capture attempts at uploading malicious or unauthorized files.

2. **Data Logging and Behavioral Analysis** All interactions with the honeypot components will be recorded in a backend database. Logged metadata will include the attacker's IP address, request payload, timestamp, request headers, and other relevant data points. This information will be used for behavioral analysis and threat profiling.
3. **Controlled, Simulated Environment** The honeypot system operates entirely within a sandboxed web environment. No real backend functionality or user data is exposed, ensuring that all activity remains isolated, non-destructive, and secure.
4. **Ethical Compliance** The project adheres strictly to ethical standards in cybersecurity research. It does not initiate interaction with external actors, nor does it provoke or manipulate attackers. Its sole purpose is observation and logging for academic and defensive purposes, without exposing any real users or live systems.
5. **Scope Limitations** The study excludes network-level honeypots, low-interaction honeypots, and integration with external intrusion detection systems (IDS). The focus is strictly on web application-level deception and its effectiveness in detecting and analyzing unauthorized access attempts.

1.5 Significance of the Study

In today's rapidly evolving digital landscape, cyberattacks have become not only more frequent but also increasingly sophisticated. Traditional security measures—such as firewalls, access controls, and input validation—are often reactive in nature and insufficient against advanced, targeted threats. As attackers continuously develop new techniques to exploit web applications, there is a critical need for security strategies that are proactive, adaptive, and intelligence-driven.

This study is significant as it investigates the implementation of web-based honeypots as an innovative and strategic approach to cybersecurity. Unlike conventional defenses that focus solely on prevention, honeypots provide a dual function: they serve as early warning systems and as tools for gaining valuable insights into attacker behavior. By embedding realistic decoy elements within a simulated web application, the system developed in this study enables the detection, monitoring, and analysis of intrusion attempts in real time—without compromising real data or functionality.

The findings of this study contribute to the growing field of cyber deception and can inform the development of more resilient web application defenses. It provides a practical framework for integrating deceptive components into security architectures, thereby enhancing situational awareness, improving threat response capabilities, and supporting continuous learning from real-world attack patterns. Ultimately, this research underscores the importance of shifting from purely reactive defense models to more intelligent and proactive cybersecurity solutions.

CHAPTER TWO

LITERATURE REVIEW

2.1 Overview of Cybersecurity Threats

As our daily lives become increasingly tied to digital platforms, web applications have become a favorite target for cyber attackers. These threats are no longer simple or random—they've grown in both complexity and scale. Hackers now routinely exploit common features like login pages, search bars, file upload tools, and APIs to find weak spots. Attacks like brute-force login attempts, SQL Injection (SQLi), and Cross-Site Scripting (XSS) are frequently used to steal data, disrupt systems, or gain unauthorized access. Malicious file uploads are also a growing concern, as attackers can sneak harmful code into systems if uploads aren't properly checked.

What makes these threats even more dangerous today is automation. With tools like Burp Suite and sqlmap, attackers can scan thousands of sites in minutes, looking for vulnerabilities—especially in areas like APIs, which often lack strong security controls. On top of that, more advanced threats like botnets, APTs (advanced persistent threats), and zero-day exploits are making it harder than ever for organizations to detect and stop attacks in time.

The traditional approach—relying solely on firewalls or antivirus software—isn't enough anymore. That's why many organizations are turning to smarter strategies. One of the most promising is the use of honeypots: fake components within a web application designed to trick and trap attackers. Instead of just blocking threats, honeypots watch how attackers behave, capturing valuable data that can help improve security moving forward.

In short, modern cybersecurity is no longer just about building strong defenses—it's about learning from the threats themselves. By studying how attackers operate, organizations can better prepare, respond faster, and build web systems that don't just survive attacks but grow stronger from them.

2.2 Introduction to Honeypots

Honeypots are a smart, proactive tool in cybersecurity that do more than just block threats—they trick and observe attackers. Instead of focusing only on keeping bad actors out, honeypots intentionally invite them in by mimicking real parts of a system, like a login page or API endpoint. But behind the scenes, these elements are fake—designed solely to watch, learn from, and log attacker behavior (Kumar & Zhang, 2022).

At their core, honeypots are decoys. When an attacker interacts with one—say, by entering credentials on a fake login form or uploading a file—the system records everything: IP addresses, browser info, payloads, and timestamps. These traps don't give access to anything sensitive but can reveal how attackers think and what tools they use (Rahman & Malik, 2024).

There are two main types:

- **Low-interaction honeypots** simulate limited parts of a system and are safer and easier to manage.
- **High-interaction honeypots** are more realistic, allowing deeper engagement, which gives richer data—but they need to be carefully isolated to avoid risk (Chen et al., 2023).

In web applications, honeypots have become more creative. Developers can hide them in plain sight, like a fake /admin-login, a pretend /api/deleteUser, or an upload form that looks real but doesn't actually store files. Since bots and scanners often go after these areas, they make excellent bait (Ibrahim et al., 2025).

Using honeypots offers several big advantages:

- **Threat Intelligence**: They collect real-world attack data to improve future defenses.
- **Early Detection**: Because no legitimate user should access a honeypot, any interaction is a red flag.
- **Efficiency**: Unlike systems that monitor all traffic, honeypots focus only on suspicious activity, reducing noise and false alarms.

- **Behavioral Insight**: They show how attackers operate—what tools they use, what they target, and how persistent they are.

2.3 Honeypots in Web Application Security

As web applications continue to grow in complexity and exposure, they've become prime targets for cyberattacks. Traditional security tools like firewalls and antivirus software often struggle to detect sophisticated, automated, or zero-day attacks. This is where honeypots step in—not as a replacement, but as a clever and proactive layer of defense. Instead of simply blocking threats, honeypots deceive attackers, drawing them into carefully designed traps that look real but are completely safe (Kumar & Zhang, 2022).

Web applications are especially vulnerable because they rely heavily on user input and are always accessible online. Attackers often exploit features like login forms, search fields, file uploads, and APIs to carry out brute-force attacks, SQL injections, XSS scripts, or malicious file uploads. By embedding honeypots directly into these features, organizations can detect suspicious activity early, gather valuable data, and analyze the attacker's behavior without risking real systems (Rahman & Malik, 2024).

For instance, a fake admin login page can mimic a real one—responding with a simple "Access Denied" while quietly logging the attacker's IP, input values, and request details. Similarly, honeypot API routes like /api/deleteUser or /api/exportDB may appear legitimate to scanning tools like sqlmap or Burp Suite but serve only to collect data on attempted exploitation (Chen et al., 2023).

Even common elements like search bars can be weaponized. A honeypot labeled "Search Users" might detect SQLi or XSS payloads and return vague errors like "Database error: near 'OR'," baiting the attacker to keep digging. Behind the scenes, every keystroke and query is recorded (Ibrahim et al., 2025).

File upload honeypots are also powerful. They mimic upload features for documents or images but are designed to capture and analyze suspicious files—like malware, backdoors, or even zip bombs—without executing them (Patel & Singh, 2024).

One of the biggest advantages of web honeypots is that they only attract malicious activity. Since regular users don't interact with these elements, any engagement is inherently suspicious. This helps reduce false positives and acts as an early warning system—alerting defenders the moment attackers begin scanning or probing a system.

More importantly, honeypots provide rich threat intelligence. By tracking payloads, tools used, request headers, and timing patterns, security teams can better understand attacker behavior and feed this knowledge back into other defenses like firewalls or intrusion prevention systems. They can even identify automated tools by the digital "fingerprints" they leave behind (Owolabi et al., 2022).

When well-implemented, honeypots are low-risk and isolated from live systems. But to be effective, they must feel convincing—complete with realistic buttons, error messages, and form responses. At the same time, logging must be detailed and secure, capturing headers, cookies, referrers, and more. The golden rule is full isolation: honeypots should never be connected to actual production data or core infrastructure.

2.4 Comparison of Honeypot Technologies

As cyber threats grow more sophisticated, honeypots have become invaluable in helping security teams understand attacker behavior without risking real systems. These digital decoys don't just act as bait—they observe, mislead, and collect intelligence in a safe, controlled environment. But not all honeypots are created equal. They vary in complexity, realism, and the depth of interaction they offer, making it crucial to choose the right type for your web application's security needs.

1. Low-Interaction Honeypots: Simple and Safe

Low-interaction honeypots are the most basic form—lightweight, easy to deploy, and safe by design. They simulate minimal functionality, like a login page that always rejects input or a static banner that mimics a service. These honeypots are perfect for catching automated bots or low-effort brute-force attacks.

Because they don’t run real services, they’re secure and consume few resources. However, their simplicity can be a drawback: experienced attackers may quickly recognize them as fake. For example, a static admin login page that always returns "Invalid credentials" may attract some probes, but it won’t fool a determined hacker for long.

2. Medium-Interaction Honeypots: Engaging with a Purpose

Medium-interaction honeypots offer more depth. They mimic dynamic behaviors without actually hosting full systems. This makes them ideal for collecting richer data on how attackers interact with fake endpoints.

For instance, a file upload honeypot might accept files and store metadata for analysis without executing anything. A fake API might return plausible JSON responses to probing tools. These systems trick attackers into believing they’ve found something useful, allowing defenders to log the payloads, techniques, and access patterns used.

They strike a smart balance: more convincing than low-interaction honeypots, yet still relatively safe and isolated.

3. High-Interaction Honeypots: Real Systems, Real Insight

High-interaction honeypots take realism to the next level. These are fully functional environments—like web servers, content management systems, or even entire operating systems—set up in isolated containers or virtual machines. They let attackers explore, exploit, and execute commands as if they were inside a real system.

Because of this deep access, high-interaction honeypots provide unmatched insight into attacker behavior. Security teams can observe not only the initial attack but also what happens next: file system exploration, lateral movement, privilege escalation, and more.

However, these honeypots carry higher risks. They require strong isolation, constant monitoring, and tight control to prevent attackers from using them as a launchpad. Still, when managed correctly, they offer a goldmine of threat intelligence.

2.5 Related Works and Gaps

Over the past decade, honeypots have carved out a respected space in the field of cybersecurity—particularly as tools for studying attacker behavior and strengthening defenses. Much of the earlier research focused on large-scale network and enterprise infrastructure, where analysts could observe widespread attacks and collect detailed threat intelligence. However, there remains a noticeable gap in research specifically focused on honeypots designed for web applications**,** especially those that mimic realistic user-facing elements like login forms, API endpoints, search bars, or file uploads.

For example, Kumar and Zhang (2022) conducted a comprehensive study on high-interaction honeypots within enterprise networks. Their work revealed valuable insights into attacker strategies post-intrusion—like privilege escalation and lateral movement. Yet, their research was largely infrastructure-centric, overlooking the web layer, where many modern attacks like credential stuffing, API abuse, and XSS take place.

On the other end, Nguyen and Okafor (2023) explored a simple web-based honeypot that imitated a login page to attract automated bots. It effectively captured a range of brute-force attempts, demonstrating how even basic web forms are routinely targeted. However, the setup was limited in scope. It didn't replicate other common web app elements like APIs or file upload interfaces, which are equally attractive to attackers.

Ibrahim et al. (2025) introduced a novel idea: a dynamic honeypot capable of adapting its behavior based on attacker input—mimicking a real system that learns and evolves. While the concept was promising and futuristic, the practical implementation was another matter. It required significant computing resources and was tested in tightly controlled environments, making it unsuitable for widespread deployment in live web systems.

Rahman and Malik (2024) made strides with a static honeypot focused on admin login portals. Their system was effective at logging credential-stuffing attempts and offered a clear window into brute-force patterns. However, it lacked broader scope—it didn't incorporate other critical web features like fake search boxes or upload fields, which limited its ability to detect more complex, multi-stage attack chains.

CHAPTER THREE

DESIGN AND METHODOLOGY

3.1 Design Goal

This section outlines the core goal of the system and provides a visual representation of how the honeypot interacts with potential attackers.

3.1.1 Design Objective

The primary objective of the honeypot-based web application is to act as a decoy system that convincingly mimics a legitimate administrative interface. This simulated environment includes key elements commonly targeted by attackers, such as:

- **Login pages** (e.g., admin login panels),
- **Search forms** that seem connected to a real database,
- **Exposed API endpoints** appearing to offer access to backend resources, and
- **File upload components** that may appear vulnerable to injection or malware-based attacks.

The core idea is to lure unauthorized users or automated bots into interacting with these fake elements under the belief that they are accessing a real, functional system. By engaging with the honeypot, the attackers unknowingly reveal valuable data points such as:

- IP addresses and geolocation,
- Attack patterns (e.g., brute-force, SQL injection, XSS),
- Payloads and scripts used,
- Time and frequency of attacks, and
- Tools or frameworks used by the intruder (via User-Agent strings or request headers).

The honeypot operates under the principle of non-interference**,** meaning it does not perform any real administrative or database operations. Instead, it logs and stores every interaction for further analysis. To prevent any real system compromise, the honeypot is isolated from critical

infrastructure**,** hosted on a separate server or sandboxed environment, ensuring it cannot be used as a pivot point for lateral attacks into the actual network.

The ultimate goals of this honeypot system are:

- To **detect early intrusion attempts**,
- To **study attacker methodologies** and motives,
- To **gather threat intelligence** that can improve overall security posture, and
- To **train machine learning models or rule-based engines** for intrusion detection/prevention systems (IDS/IPS) using real-world data.

This proactive approach enables system administrators and cybersecurity professionals to gain a deeper understanding of real threats in a controlled, low-risk setting—transforming attempted attacks into useful data for defending actual assets.

3.1.2 System Flow Overview

The operational flow of the honeypot-based web application is designed to simulate a realistic web environment capable of engaging, logging, and analyzing malicious behavior. The system follows a structured progression from attacker interaction to data-driven security insight, involving multiple fake modules and deception layers. This flow ensures comprehensive monitoring without exposing any actual assets.

1. Honeypot Discovery

The process typically begins when a potential attacker—or an automated scanner—discovers the web application through methods such as web crawling, brute-force scanning, or probing commonly used endpoints like /admin, /api/v1, /search, or /upload. These endpoints are deliberately exposed to attract interest by simulating the presence of an active administrative or backend interface.

2. Attacker Engagement

Upon reaching the honeypot, the attacker interacts with one or more simulated components of the web application. These components are carefully designed to mimic real functionalities while capturing the attacker's behavior. The system includes:

- A **fake login page** that accepts input but is never authenticated, intended to detect brute-force or credential stuffing attacks.
- **Simulated API endpoints** that respond to automated probing or malformed requests to study API abuse and exploitation attempts.
- A **search form** that quietly records injected SQL or JavaScript strings to analyze SQL injection (SQLi) and cross-site scripting (XSS) behaviors.
- A **file upload interface** that accepts file submissions and isolates them for inspection, enabling detection of malware or shell upload attempts.

3. Logging and Deceptive Response

All user actions are monitored in real time and silently logged by the backend system. Information such as request payloads, IP address, geolocation, HTTP headers, user-agent strings, and timestamps are captured for every interaction. To maintain attacker engagement, the honeypot generates realistic but fabricated feedback, such as:

- "Invalid username or password" messages from the login form.
- Standard HTTP status codes (e.g., 401 Unauthorized, 404 Not Found) or mock API responses.
- Fake search result pages with no actual database connection.
- Confirmation messages like "File uploaded successfully," even though the upload is sandboxed and never executed.

4. Data Storage and Classification

Captured data is persistently stored in a MongoDB database using a flexible schema. Each record is enriched with metadata and tagged according to its corresponding attack vector—such as brute-

force, XSS, SQLi, or file-based malware. Uploaded files are securely sandboxed for later static or behavioral analysis to identify potential threats without compromising system integrity.

5. Analysis and Threat Intelligence

In the final phase, security analysts or automated detection systems review the accumulated data to derive actionable insights. This includes identifying frequently used attack techniques, tracking persistent attackers based on behavioral patterns, and measuring the effectiveness of the honeypot's deception strategies. The resulting intelligence contributes to refining defensive mechanisms, tuning intrusion detection systems (IDS), and informing broader cybersecurity policies.

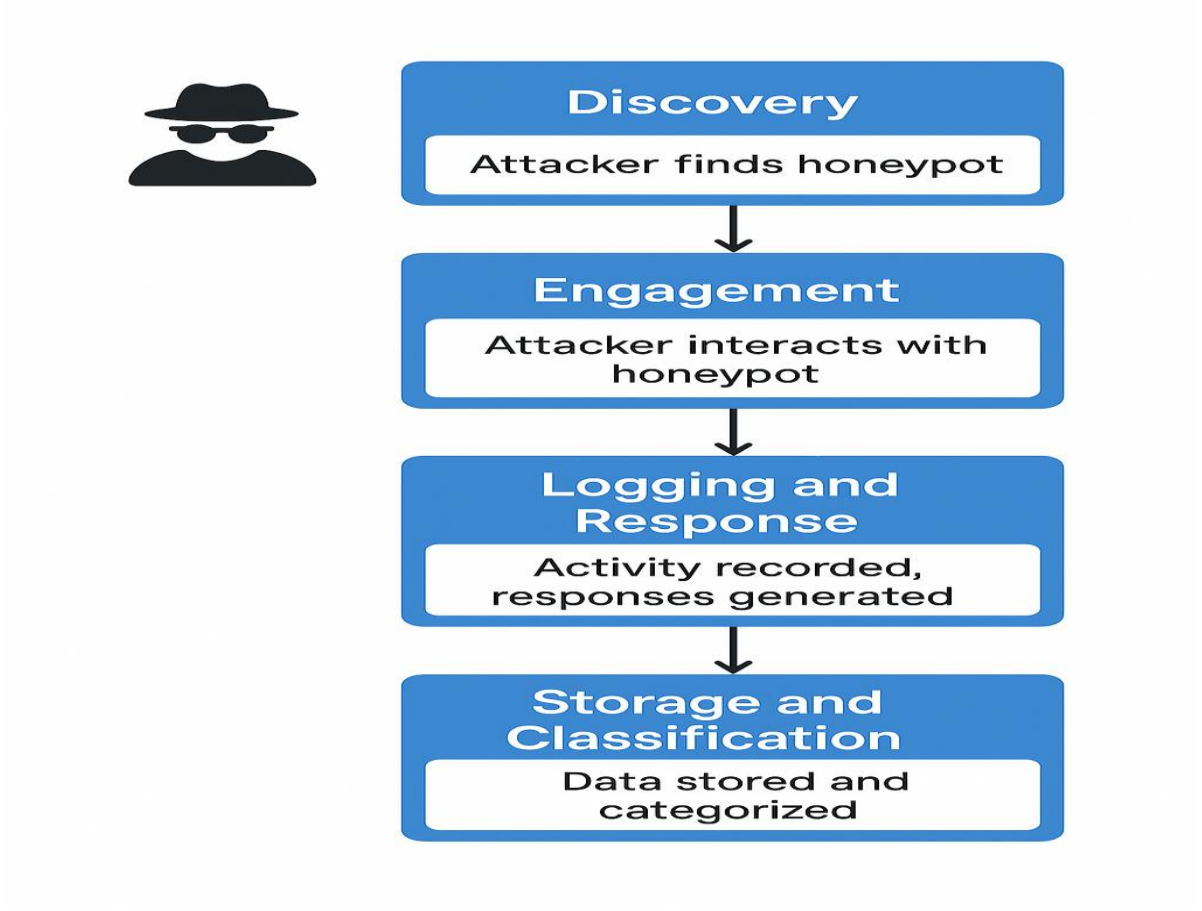

Figure 3.2: HONEYPOTS ATTACK SYSTEM FLOW

3.3 The Proposal Architecture

Designing a honeypot web application is like building a house that looks real from the outside—but inside, it's all a carefully crafted illusion. The goal is to make attackers believe they've found a vulnerable system, while in reality, every move they make is being monitored, logged, and analyzed in a secure and isolated environment.

To achieve this, the system is structured using a three-tier architecture**,** consisting of the Presentation Layer**,** Application Logic Layer**,** and Data Storage Layer**.** This layered approach ensures the system is modular, scalable, and easy to maintain, all while keeping the actual environment safe from harm.

At the heart of the architecture is the idea of deception through realism**.** The honeypot mimics a typical web application, like a CMS dashboard or admin panel, complete with login forms, API endpoints, search fields, and file upload options. But none of these functions are real. They're traps—designed to engage attackers, make them believe they're making progress, and in the process, reveal their tactics and tools.

3.3.1 Overview of System Components

A. Presentation Layer (Frontend): This is the interface that attackers interact with. It is carefully designed to resemble a legitimate admin dashboard. The layer includes a login form that records brute-force attempts, a fake dashboard that users are redirected to after a "successful" login, and a search bar intended to simulate database interactions and entice attackers into testing SQL injection or XSS payloads. Additionally, a file upload form is available, which accepts potentially malicious files for later analysis. To maintain credibility, the interface provides realistic UI feedback such as "Login failed" or "File uploaded successfully." The primary objective of this layer is to convince attackers that they are interacting with a genuine application, keeping them engaged and active within the honeypot.

B. Application Logic Layer (Backend): The backend is responsible for handling all interactions behind the scenes. It logs every action performed by the attacker, including form submissions, uploads, and API requests. It captures essential metadata such as IP addresses, headers, and

timestamps, and employs pattern-matching techniques to detect common attack types like SQL injection and cross-site scripting. The backend also generates fake responses to maintain the illusion of a real system, without exposing any functional logic. For instance, login failures and unauthorized API messages are simulated. It also tracks sessions to monitor attacker behavior over time. Importantly, all uploaded files are stored in a sandboxed environment and never executed. The system processes no real data or business logic, ensuring a secure operating environment.

C. Data Storage Layer (Database): This layer utilizes MongoDB to store all collected interaction data. It records login attempts, payloads, file metadata, and API traffic. Entries are categorized with threat tags such as "SQLi," "XSS," or "File Upload" to facilitate filtering and analysis. Indexed fields like IP addresses and endpoint paths support efficient querying, enabling security teams to identify specific incidents or trends. The schema-less structure of MongoDB makes it flexible and adaptable for various data types, supporting both real-time monitoring and long-term threat analysis.

3.3.2 Logical System Flow

The system follows a structured flow from attacker interaction to data capture and analysis. Initially, an attacker discovers the honeypot, often through automated scanning or targeted probing of URLs like /admin or /cms-login. Once engaged, the attacker attempts actions such as logging in, accessing API endpoints, performing searches, or uploading files. Each of these interactions is silently recorded by the backend, along with relevant metadata. In response, the system provides convincing but fabricated feedback to keep the attacker engaged without revealing its true nature. All collected data is then stored in MongoDB, where security analysts can later review logs, analyze patterns, and extract valuable intelligence from the captured behavior.

3.3.3 Security and Isolation

To prevent any actual system compromise, the honeypot is designed with strict security and isolation measures. No real user data or application logic is exposed to attackers. All uploaded files are safely stored without ever being executed, ensuring that malware or scripts cannot harm the system. The entire honeypot runs in an isolated environment, such as a Docker container or a

sandboxed virtual machine, which further limits any risk of lateral movement or unauthorized access. Logging is handled asynchronously, which not only improves system performance but also ensures resilience under heavy traffic or during ongoing attacks. This allows the honeypot to continue operating effectively, even when actively targeted.

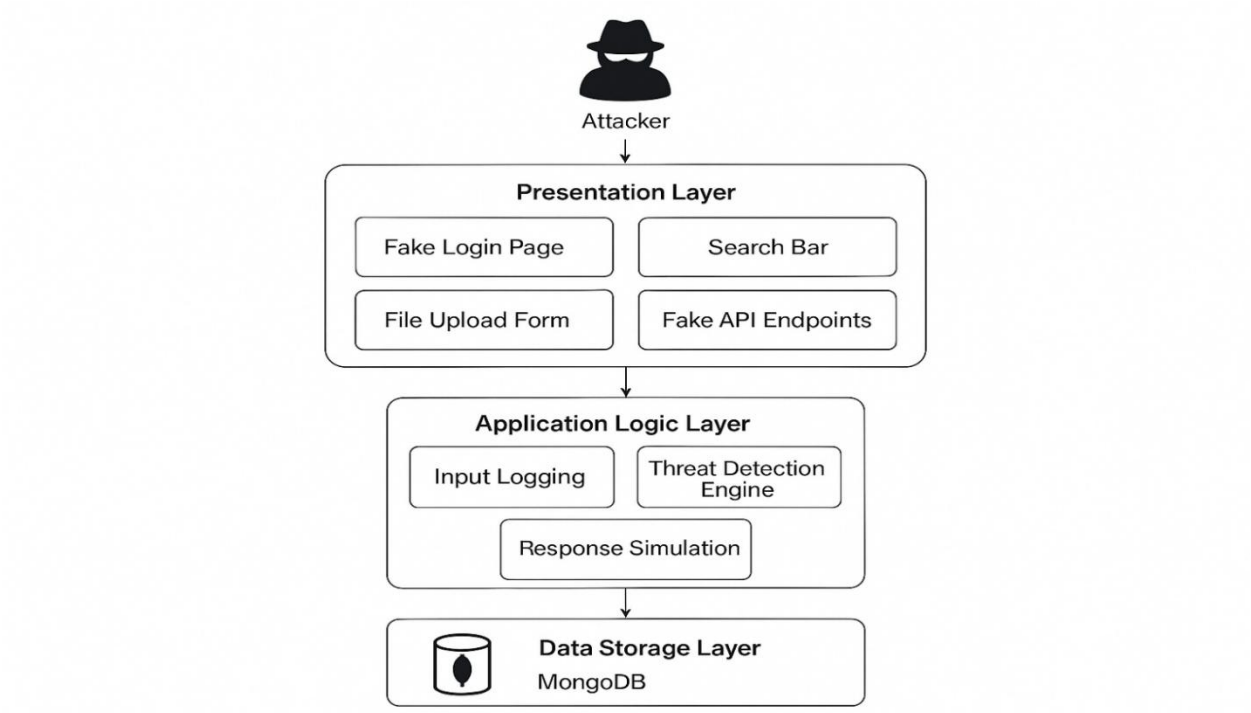

Figure 3.3: HONEYPOTS ATTACK SYSTEM ARCHITECTURE

3.4 The Use Case Diagram

In this honeypot web application, the primary "user" is an unauthorized actor—typically a hacker, bot, or automated scanner. The system is intentionally designed to deceive and observe, not to serve legitimate users. It mimics real application behavior to encourage interaction while secretly capturing attacker data for analysis.

Key Use Cases:

1. **Attempt Login** The attacker tries to log in through a fake admin page. Every attempt is logged, helping detect brute-force patterns and common credentials.

2. **Access Fake API Endpoint** The actor sends requests to sensitive-looking API routes. Fake responses are returned while all request data is recorded to understand attack tools and methods.
3. **Submit Search Input** Suspicious or malicious inputs in the search form (like SQLi or XSS) are logged. Realistic error messages are shown to keep the attacker engaged.
4. **Upload File** Malicious files (e.g., shells or scripts) uploaded through a fake form are sandboxed and logged. This helps detect file-based attacks or RCE attempts.
5. **Trigger Error Simulation** The system simulates believable error messages when attackers send malformed or probing inputs. This enhances realism and captures attacker behavior.

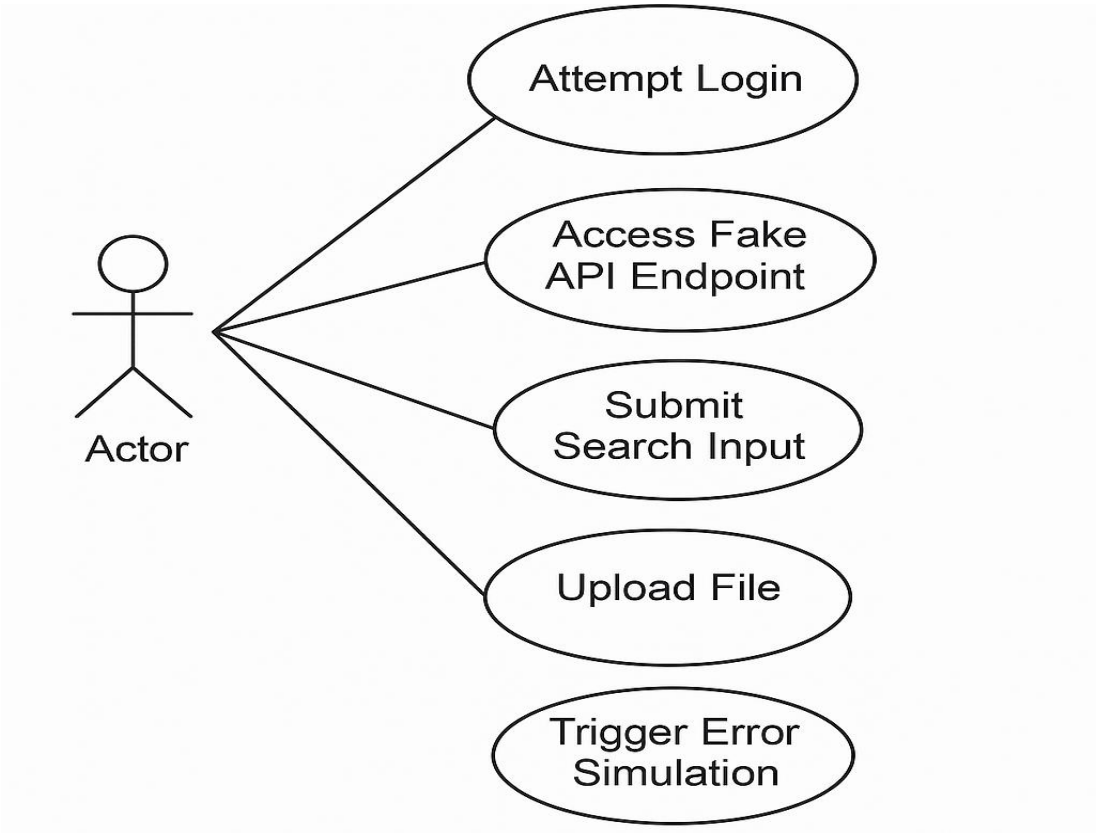

Figure 3.4: HONEYPOTS ATTACK USE CASE DIAGRAM