International Journal of Emerging Technology in Computer Science & Electronics (IJETCSE)
ISSN: 0976-1353 Volume 32 Issue 01 – JANUARY 2026.

Detection and assessment of Distributed Denial of Service Attacks using honeypots

Bhavya A.R[#1], Nagamma[#2]

CSE, AKASH INSTITUTE OF ENGINEERING AND TECHNOLGOY, DEVANAHALLI, BANGLORE, INDIA

CSE, AKASH INSTITUTE OF ENGINEERING AND TECHNOLGOY, DEVANAHALLI, BANGLORE, INDIA

Abstract—
This project addresses the problem of safely detecting and analysing malicious login attempts and common web attacks such as brute force, SQL injection, and XSS on a web application. It does this by using a realistic Instagram-like fake login page as a honeypot, which captures all login attempts along with IP address, username, password, and time, then classifies them as normal or different attack types using predefined detection rules. The project work carried out includes designing the phishing-style frontend UI, developing the Flask backend with attack detection logic, creating a SQLite database for structured logging, integrating Telegram bot and Twilio SMS for real-time alerts, and implementing a secured dashboard accessible only with special credentials. The dashboard displays the collected attack data in an Excel-style table and visualizes the distribution of attacks with charts (bar or doughnut) showing counts and percentages, making analysis easier.

The outcome of the project is a fully functional web-based honeypot and monitoring system that demonstrates how attackers can be trapped using a familiar UI and how defenders can log, classify, and visualize suspicious activities for awareness and research. It provides a practical learning platform for web security, phishing risks, logging, alerting, and basic threat intelligence generation. Future work can extend this system into a richer honeynet by adding more fake pages and services, including advanced detection for other attack vectors (command injection, directory traversal, credential stuffing), integrating IP geolocation and world maps, exporting logs for SIEM or machine learning analysis, and deploying the solution on a public cloud server to observe real-world attack traffic over a longer period.

Index terms — DDOS attack, Honepots, Attacks, Python, Phishing, Security, Networking

I. INTRODUCTION

The proposed project is a honeypot-based “Instagram-style” web security system designed to attract, detect, and analyze malicious login activity in a safe, controlled environment. It presents users with a fake social media login page that closely resembles a real Instagram login, encouraging attackers to attempt credential guessing or inject malicious payloads. Behind this realistic interface, a Flask-based backend captures every login attempt, recording details such as IP address, username, password, timestamp, and a classified attack type (Normal, Brute Force, SQL Injection, or XSS) in a structured SQLite database.

Not just about catching intruders, the setup keeps watch using layers - spotting threats, sounding alarms, then showing what happened. When one address tries too many times, custom rules catch it; sneaky code attempts get caught by pattern checks in form inputs. A warning jumps out instantly through Telegram or SMS once something odd shows up. Only admins see the control screen, where rows of data sit like spreadsheets beside live graphs built with Chart.js, mapping how attacks spread and shift over time. That mix turns it into more than software - it teaches how break-ins work, how traps respond, how defences learn.

II. RELATED WORK

This project lives in cybersecurity, focusing on web app protection along with honeypot setups. Its niche? Web security mechanisms paired with deceptive trap systems meant to catch attackers off guard

Tricking hackers often means setting up something that looks real but isn’t. A pretend login page, much like Instagram, can draw them in without warning. Once they bite, everything slows down - movement watched, every click noted. Instead of stopping at the door, let them walk right through. What happens next reveals how they think. Fake setups act like quiet observers, hidden in plain sight. Safety comes from distance, not confrontation. Watching teaches more than blocking ever could.

A fresh approach begins here - building a full system online that looks like Instagram's sign-in screen. This setup pulls in suspicious visitors by pretending to be real. Watching them happens safely inside locked conditions. Every step, from drawing plans to making it work, gets handled carefully. Instead of rushing, each piece fits slowly into place. The goal stays clear throughout: learn how intruders behave when they think no one sees.

Starting with Python and Flask, the system builds backend processes that respond to sign-in attempts. When someone tries to log in, it checks for suspicious behavior like repeated failures. Patterns tied to brute force attacks get flagged automatically. If signs of

Web attack detection: Focusing on identifying brute force, SQL injection, and XSS attacks at the login layer using input inspection and IP-based rate analysis.

Security monitoring and incident response: Logging all attempts in a database, generating real time alerts via messaging and SMS, and providing a dashboard for security analysis and decision making.

Educational / research security tools: Serving as a teaching and experimentation platform to study attacker patterns and demonstrate practical web security concepts using a full-stack implementation.

III. Existing System

A trap hidden in plain sight - that is what this setup becomes for anyone probing too hard. Instead of just locking doors, it watches hands reaching for the knob. Fake login pages shaped like familiar apps draw in those who should not be there. Every keystroke typed by intruders gets recorded - time, IP, password choice - all stored cold and clear. Unlike heavy defenses slowing down real users, this one sits quiet until something stirs. It speaks only when needed, sending word through messages the moment suspicion spikes. Real usernames stay locked away while impostor attempts pile up behind glass. Charts grow quietly in the background, showing rhythms of attacks across days. Rows of logs display where each try came from, painting maps without labels. Tools buried inside spot known tricks: forced entries, script injections, sneaky redirects. Nothing runs live beyond the decoy; no risk spreads to actual servers. Alerts land fast - not later, not maybe - but now, straight to phones and screens. The whole thing installs without drama, works alone, adapts silently. Watching instead of shouting, learning more with every failed breach.

IV. Proposed System

The honeypot-based “Instagram-style” security project has A realistic-looking fake login screen appears, built to mirror an actual social media sign-in. Its layout copies Instagram closely, aiming to trick those with bad intentions. The setup pulls attention through familiar colors and spacing. Every detail follows common patterns people expect when logging in. It operates quietly, waiting for someone up to no good to step forward. Visual cues match what users see daily online. This mimicry works because it feels normal at first glance. Trust builds quickly through resemblance, not promises. Deception hides in plain sight, shaped like something harmless.

The honeypot-based “Instagram-style” security project has the following objectives:

A realistic-looking fake login screen appears, built to mirror an actual social media sign-in. Its layout copies Instagram closely, aiming to trick those with bad intentions. The setup pulls attention through familiar colors and spacing. Every detail follows common patterns people expect when logging in. It operates quietly, waiting for someone up to no good to step forward. Visual cues match what users see daily online. This mimicry works because it feels normal at first glance. Trust builds quickly through resemblance, not promises. Deception hides in plain sight, shaped like something harmless.

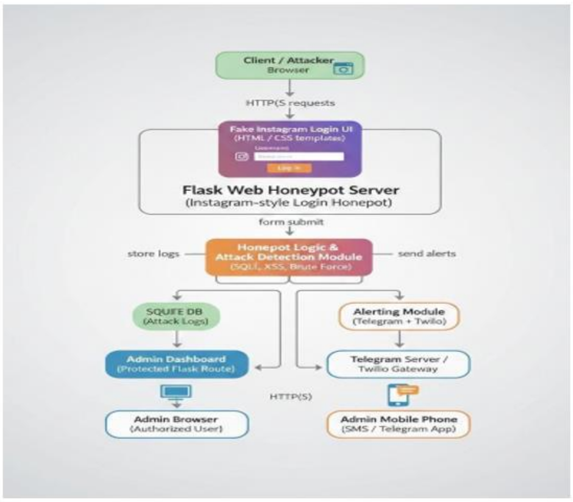
Client / Attacker Browser
HTTP(S requests
Fake Instagram Login UI (HTML / CSS templates)
Flask Web Honeypot Server (Instagram-style Login Honepot)
form submit
store logs
Honepot Logic & Attack Detection Module (SQLI, XSS, Brute Force)
send alerts
SQUITE DB (Attack Logs)
Alerting Module (Telegram + Twilo)
Admin Dashboard (Protected Flask Route)
Telegram Server / Twilio Gateway
HTTP(S)
Admin Browser (Authorized User)
Admin Mobile Phone (SMS / Telegram App)

Fig 1: Data Flow Diagram

V. Methodology

The Deception Front-End

The process begins when a **client or Attacker** accesses the system via a web browser. The **Flask Web Server** serves a "Fake Instagram Login UI." This is a social engineering trap designed to look indistinguishable from the real Instagram login page. Because it is a honeypot, any interaction with this page is inherently suspicious, as no legitimate users should be directed here.

Attack Detection & Processing

Once a user submits a form, the data is passed to the **Honeypot Logic & Attack Detection Module**. Unlike a real login page that verifies credentials, this module inspects the payload for common web vulnerabilities:

SQL Injection (SQLi): Detecting attempts to bypass the login via database queries.

Cross-Site Scripting (XSS): Identifying malicious scripts injected into input fields.

Logging and Management

The system handles the captured data through two channels:

SQLite DB: A lightweight database that stores "Attack Logs," providing a historical record of every interaction for forensic analysis.

Admin Dashboard: A secured, authenticated Flask route that allows the system administrator to visualize attack trends and view logs in a human-readable format.

Real-Time Alerting Pipeline

To ensure immediate response, the system utilizes an **Alerting Module** integrated with third- party APIs:

Telegram/Twilio: The system uses the Telegram Bot API and Twilio Gateway to push instant notifications.

End-to-End Notification: The administrator receives an SMS or a Telegram message on their **Mobile Phone** the moment a high-severity attack is detected, allowing for real-time monitoring of the threat landscape.

VI. Module Description

Start and Initialize System

Start the Flask web application.

Configure the SQLite database and create the Attack Log table with fields: id, IP, username, password, time, attack_type.

Initialize an in-memory attempt tracker map to store recent login times per IP.

Set brute force parameters BRUTE FORCE THRESHOLD and BRUTE FORCE TIME WINDOW.

Configure Telegram bot token and chat ID.

Configure Twilio account SID, auth token, Twilio phone number, and defender mobile number.

Display Honeypot Login Page

• When the user sends a GET request to /, render the Instagram-style login page with normal UI (no trap message).

Handle Login Submission (Honeypot Logic)

When a POST request is sent to /, read username and password from the form and get the client IP and current time.

If username=="BHARATH" and password=="Amol&12485", treat it as a valid admin login and redirect to the /dashboard page.

Otherwise, update the attempt tracker for this IP:

Remove timestamps older than BRUTE_FORCE_TIME_WINDOW.

Add the current timestamp to the list for this IP.

Initialize attack type as "Normal".

Generate and Send Alerts

Build a text message including IP address, username, time, and (if any) detected attack type.

Send the message to Telegram using the Telegram Bot API for every login attempt.

If attack_type is not "Normal", send an SMS alert using Twilio to the configured defender phone number.

Log Attempt to Database

Insert a new record into the Attack Log table with IP, username, password, time, and attack_type.

Render Response Page

If attack_type is "Normal", re-render the Instagram-style login page so the attacker sees a normal failure.

If attack_type is "Brute Force Attack", "SQL Injection", or "XSS Attack", render the full-screen trap view that shows “You got trapped” with the laughing emoji on the gradient background.

Dashboard View (Admin Only)

When a GET request is sent to /dashboard, query the database to get:

Latest 100 attack logs.

Total number of logged attempts.

Number of unique IPs.

Counts of each attack type (Brute Force, SQL Injection, XSS).

Render the dashboard page with:

An “Attack Summary” section showing the counts.

An Excel-style HTML table showing IP, username, password, time, and attack type for each log entry.

A Chart.js doughnut chart that visualizes the counts of each attack type and displays percentages for each slice.

VII. CONCLUSIONS

The honeypot-based “Instagram-style” security project successfully shows how a realistic decoy web application can be used to safely study attacker behavior, detect common web attacks, and generate useful security insights in real time. By imitating a familiar social media login interface, the system attracts attackers without exposing real user accounts or production systems, and logs every interaction with IP address, username, password, timestamp, and attack classification into a structured SQLite database. This approach follows established honeypot principles: the goal is not just to trick attackers, but to observe and learn from their techniques to improve security.

From an implementation perspective, the project delivers a complete full-stack solution using a compact and practical technology stack. Python with Flask handles routing, input processing, attack detection, database operations, and integration with external messaging services, while HTML and CSS provide a convincing fake login UI, trap screen, and a clean admin dashboard. JavaScript with Chart.js adds interactive charts that visualize the distribution of brute force, SQL injection, and XSS attacks, turning raw logs into easily understandable patterns. Custom detection logic tracks repeated attempts from the same IP to detect brute force and uses regular expressions to flag SQL injection and XSS payloads, covering three major web attack categories. Each classified attempt triggers immediate notifications via messaging and SMS, tying detection, logging, and alerting into a lightweight monitoring pipeline suitable for labs and small environments.

In conclusion, the project meets its objectives by providing a realistic decoy login environment, robust attack detection, comprehensive logging, real-time alerting, and clear visualization in a simple, deployable design. It demonstrates how modest tools can be combined into a practical cybersecurity and educational platform, reinforcing key concepts such as honeypots, web attack detection, monitoring, and incident awareness, while leaving room for future extensions like additional attack signatures or integration with larger security monitoring systems.

Support for additional attack types
Extend detection beyond brute force, SQL injection, and XSS to include patterns like command injection, directory traversal, CSRF indicators, and credential stuffing so the honeypot can capture a wider range of real-world attacks.

GeoIP and location-based insights
Integrate IP geolocation to display attacker countries/regions and simple maps or country-wise statistics on the dashboard, helping to understand geographical trends in malicious activity.

Integration with external security tools
Add options to export logs (CSV/JSON/syslog) or send events to external security platforms (e.g., SIEM/SOC tools), allowing the honeypot data to be correlated with other security logs in larger environments.

Scalable multi-instance honeynet
Deploy multiple honeypot instances (different ports, domains, or containers) and aggregate logs into a central dashboard, effectively turning the system into a small honeynet with broader visibility.

Advanced reporting and analytics
Provide downloadable reports (e.g., Excel or PDF) summarizing attack volume, time- based trends, top attacking IPs, and most frequent payloads, supporting periodic security reviews and presentations.

Role-based and secure admin access
Replace single hardcoded admin credentials with proper authentication and role-based access (admin, read-only analyst), improving security and making the dashboard usable by multiple team members.

Operational hardening and deployment improvements
Harden the deployment with HTTPS, reverse proxies, containerization (e.g., Docker), and environment-based configuration so the honeypot is safer to run in lab or cloud environments and easier to move between systems.

REFERENCES

[1] M. Abu Ahad *et al.*, "Dynamic Interactive Honeypot for Web Application Security," *International Journal of Wireless and Mobile Computing*, vol. 14, no. 6, 2024. [Online]. Available: https://www.mecspress.org/ijwmt/ijwmt-v14-n6/IJWMT-V14-N6-1.pdf

[2] M. Nunes, P. Figueiredo, and N. Neves, "Web Application Risk Awareness with High Interaction Honeypots," in *Proc. INForum – Simpósio de Informática*, 2010. [Online]. Available: https://www.dpss.inescid.pt/~mpc/pubs/nunes-inforum10.pdf

[3] "Dynamic Interactive Honeypot for Web Application Security (HTML Version)," *International Journal of Wireless and Mobile Computing*, 2024. [Online]. Available: https://www.mecs-press.org/ijwmt/ijwmt-v14n6/v14n6-1.html

[4] "From Risk Awareness to Security Controls: Benefits of Honeypots to Web Application Security," *International Workshop on Business Applications of Security (IBWAS)*, 2010. [Online]. Available: https://repositorio-cientifico.essatla.pt/bitstream/20.500.12253/537/1/IBWAS2010.pdf

[5] M. A. F. Noor *et al.*, "Securing Web Applications Against XSS and SQL Injection Attacks Using a CNN– LSTM Deep Learning Model," *Applied Sciences*, 2024. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC10799887/

[6] P. K. Sahu and S. S. Behera, "An Approach to Detect and Prevent SQL Injection and XSS Attacks," *International Journal of Engineering Trends and Technology*, vol. 71, no. 8, 2023. [Online]. Available: https://ijettjournal.org/Volume-71/Issue-8/IJETT-V71I8P219.pdf

[7] S. R. Pawar and P. K. Shukla, "An Approach for Detecting and Preventing SQL Injection Attack Using Query Sanitization," *International Journal of Computer Trends and Technology*, vol. 49, no. 4, 2017. [Online]. Available: https://www.ijcttjournal.org/Volume49/number-4/IJCTT-V49P139.pdf

[8] A. Mookhey, "Detection of SQL Injection and Cross-site Scripting Attacks," *Black Hat USA Whitepaper*, 2004. [Online]. Available: https://www.blackhat.com/presentations/bh-usa-04/bh-us-04-mookhey/old/bh-us04-mookhey_whitepaper.pdf