CHATGUARD : A PYTHON-BASED FAKE CHAT APPLICATION TO DETECT SOCIAL ENGINEERING BEHAVIOUR IN SHARED COMPUTING ENVIRONMENTS

Laxmi Raja[1], K Amuthabala[2], R. Srikanth[3]

ABSTRACT

Social engineering is one of the least considered yet highly critical types of cyber-attacks, especially in shared environments such as computer labs, libraries, and schools. While technical threats tend to focus on systems, social engineering leads people to disclose sensitive information inadvertently. To address this issue and foster a cybersecurity culture, a fake chat application is available, known as ChatGuard, written in a simple programming language using the Tkinter framework. It pretends to be a chat interface, secretly serving as a honeypot to identify and record any suspicious user activity based on predefined or custom keywords, such as 'password', 'OTP', or 'admin'. The software can be used as an educational tool in an Outcome-Based Education (OBE) environment, allowing students to learn and simulate basic human-target cyberattacks. It will enable role-playing, keyword substitution, and log-based tracking, promoting both technical and ethical awareness. Supposed to be run as an offline program, for personal use, on your favorite shared servers to mimic phishing attacks during training or awareness sessions. Furthermore, this paper correlates Course Outcomes (COs) with appropriate Program Outcomes (POs), including a range of skills in GUI programming, file I/O (input/output), user studies, and cybersecurity ethics. ChatGuard also encourages students to apply their basic programming knowledge responsibly, with an emphasis on the responsible use of software. With its ease of use, flexibility, and moral guidance, it provides an excellent learning resource for understanding and developing thinking around social engineering risks.

Keywords : Social Engineering, Honeypot Simulation, Cybersecurity Education, Keyword Detection, GUI Application, Offline Security Tool, Ethical Hacking, Shared Computing Environments

Department of Cyber Security[1]
Karpagam Academy of Higher Education, Coimbatore, India[1]
laxmirajaphd@gmail.com[1]

School of Computer Science and Engineering[2]
REVA University, Bangalore, India[2]
amuthabala@gmail.com[2]

Department of Computer Science and Engineering[3]
Hindustan Institute of Technology and Science, Coimbatore, India[3]
csesrikanth2010@gmail.com[3]

* Corresponding Author

I. INTRODUCTION

Security and social engineering: Cybersecurity threats evolve, as do society, technology, and wit. One of the most powerful, yet least-known, forms of attack in the digital shared space is social engineering. Social engineering differs from real hacking because it does not target confidential information stored in a computer; instead, it targets the mindset of the people it is trying to attack, including passwords and other confidential details. This type of attack is well-suited to the chained environment of a public library, an internet café, or a university computer lab where access to the network is shared [1]. The attacker can pretend to be a trusted server or individual and spoof user data. Schools are especially susceptible to these attacks, as students are often not fully informed about cybersecurity and may inadvertently be exploited or drawn into such attacks [2]. Efforts to educate and raise awareness about these human-centered threats have never been more critical for establishing sustainable digital ecosystems and protecting users.

Here, we propose ChatGuard, a lightweight false chat interface written in Python aimed at detecting early social engineering indicators through input-based keyword filtering. ChatGuard is an offline script developed using the Tkinter GUI toolkit, which utilizes a chat-based interface that logs messages containing preset and customizable suspicious keywords (e.g., OTP, admin, password) [3]. These words are also commonly used in phishing scams and manipulation attempts. The program quietly records these messages in a "hidden" file that allows an administrator to review them. This feature allows teachers or administrators to view trends in user behavior without disrupting the users. Additionally, a role-based mode (such as student or admin) allows rudimentary access simulations to stress the importance of good user authentication and trust.

The assignment is carefully crafted to meet specific Outcome-Based Education (OBE) criteria, encompassing security awareness, computing ethics, GUI programming, and file I/O. ChatGuard is designed to motivate students to apply critical thinking when it comes to protecting their virtual and physical environments, and simulates a real-world

attack vector in a safe, controlled environment. It is flexible with a customizable keyword extraction function, allowing teachers to adjust the tool to their particular teaching content or language [4]. The work also supports the infusion of ethics into technical education. The tool also augments theory-driven cybersecurity teaching by positively reinforcing the human errors that lead to security breaches, based on hands-on learning. Instructors are encouraged to "practice ethical implementation" by communicating the ethical purpose of the tool, gaining the user's consent and upholding the educational boundary in its use [12].

Ethics and privacy are essential aspects of contemporary computing education, and ChatGuard reveals this in its design. The tool was not designed for surveillance or collecting information without permission, but rather as a teaching tool for cybersecurity courses: vulnerability raising campaigns, demonstrations in ethical hacking and lab-based simulations with prior consent. With ChatGuard, students will have the opportunity to study how attackers leverage human trust and how a straightforward tool can help in defense, as part of a community of ethical practice surrounding their technical skills development. ChatGuard is a simple, flexible, and highly ethical system that can be used to teach cybersecurity in shared environments at minimal cost.

The main contributions of this work are the following :

- Developed a Python Graphical User Interface (GUI) tool for fake chatting to mimic Social Engineering attacks and prevent them in a distributed environment.
- Introduced embedding, customized keyword detection, and silent logging as a combined feature to watch for questionable user inputs without any interruption to regular activity.
- Aligned the tool with OBE-based objectives to improve awareness of cyber security with practical legal simulations.

The rest of the paper is organized as follows. We discuss related works in Section II on social engineering detection, honeypot-based security tools, and educational cybersecurity simulations. In Section III, we present the proposed approach, which comprises a system architecture, UI design using Python's Tkinter, keyword recognition logic, and considerations for privacy. In Section IV, we present implementation setup, simulation scenarios, and example results, including keyword-triggered logs and user role-based interactions. SECTION V: Conclusion This paper concludes in Section V by recapping the primary contributions and proposing future improvements such as AI-based language detection, cloud-based log monitoring, and broader classroom deployment plans.

II. LITERATURE REVIEW

Charith et al. (2025) discussed dynamic social engineering attack models using a multi-platform honeypot for awareness and defense training. Their approach focuses on user-oriented risks and shows that interactive deception is capable of mimicking authentic phishing in practice. Featuring modular and educational deployment, their system synchronizes with the ongoing development of the awareness-oriented training instruments. Their publications directly aid tools such as ChatGuard that implement a honeypot in chat, and simulate adversary interactions as if a real adversarial action occurs in an educational environment, and behave to leverage users with an efficient use of the honeypot-based system operation space. The more emphasis on simulation for behavior learning is consistent with our GUI-based and role-controlled offline design [1].

McKee, Noever (2023) explored AI-driven conversational honeypots to observe and log detrimental engagement behaviors. Their method leverages chatbot interfaces as a gateway to explore attacker psychology in text-based interactions. The paper demonstrates that realistic human-like interfaces can be used to lure and observe user tricks. This is conceptually similar to ChatGuard's fake chat interface that records inaudible keystrokes without user knowledge. The emphasis on naturalistic input and silent tracking provided the inspiration for our simulation approach [3].

Priya and Chakkaravarthy (2023) introduced a containerized honeypot deception approach in distributed cloud networks to detect the attacker's action. Although targeted to industrial systems, their focus on isolation, stealth, and local logging justify the design rationale followed a la ChatGuard. They demonstrate that modular deception can work with little overhead, in particular, when logs are kept in the local system and offline settings are used to configure them. Their approach is flexible and is compatible with the lightweight implementation of our educational tool in shared labs [5].

Abualhija et al. (2023) simulated honeypot that is specialized to protect against social engineering attacks at interaction level. Their approach works on the process of capturing and logging certain user's actions in collaborative systems, and inferring human-oriented manipulation characteristics. This concept, inspired us for the implementation of ChatGuard, that detects "porn and sex" sensitive or suspicious user input, and secretly logs it for the attention of the system administrator. Their focus on behavior

analysis complements the goal of our tool to increase awareness using role-based simulation and ethical logging [6].

Javadpour et al. (2024) conducted a thorough survey of cyber deception frameworks beginning with basic honeypots and their more advanced variants with an application of Artificial Intelligence (AI). Their work include; learning, research and real-world uses of modular honeypot systems. They emphasize the usability, ethical application, and scalability which also match with the design needs of ChatGuard. These authors' review also contextualizes the use of deception as a means not only of defense, but also of learning how to operate in such environments in a controlled setting like universities and training programs [7].

Lanka et al. (2024) have analyzed using AI models for intelligent threat detection using honeypot log analysis. Their model is user-interaction driven, in that they learn threat signatures and attack patterns based on user behavior data. While their approach is tailored for large-scale attack profiling, the philosophy of supervision from logged behavior facilitates natural extensions of ChatGuard at a later stage. Their work highlights the future roadmap using text-based keyword detection in combination with AI models for smart phishing and manipulation detection in the long term [8].

Moriæ et al. (2025) surveyed cyber security plans that utilize honeypots and unnatural resources for generating real-time use by actors. They introduce modular deception mechanisms as proactive defense squares and ethical transparency enablers in deployment. The findings in their research advocate that the role of simulated interaction for strengthening user awareness and trust-based vulnerabilities in decreasing, which is consistent with the aim of ChatGuard in educational labs. The focus on ethical design, modular extensibility, and proactive awareness raising is the key aspect informing our design of the system [4].

Nawrocki et al. (2023), based on a data-driven approach, considered the detection with a honeypot of the amplification DDoS attacks. Their approach is built on behavioral analysis, and automatic logging aligns with ChatGuard's quiet type detection at the network layer. The work also contributes to the general body of knowledge on passive monitoring in information security and verifies the applicability of low-interaction deception platforms. Their lessons learned from structured logging provide principles that are transferable to local user-input monitoring systems, such as ours [9].

III. SYSTEM ARCHITECTURE AND METHODOLOGY

The overall goal of ChatGuard is to provide an operational and ethical channel for simulating and discovering social engineering profiles by sharing computational resources. Designed with educational use in mind, the design is modular, easy to understand, and works offline. Here, we describe the system's structure, its main modules, and how they were implemented during the system's development. The method involves GUI development using Python's Tkinter, dynamic keyword filtering, log handling, and ethical functionalities, including user role imitation and interaction constraints. The architecture is composed of three segments: system flow and design, GUI with keyword detection, and the integration of ethical and instructional elements.

A. System Architecture and Flow (with Fig. 1)

Architecture: The ChatGuard is structured into three major modules: the User Interface Module, the Keyword Detection Engine, and the Logging Handler. At the start of the application, the GUI takes input through an entry field. Every message input to the system is transferred to an Input Handler, which receives each message and forwards it to the Keyword Detection Engine, which checks the message against a list of sensitive keywords [6]. If it's a match, the Logging Handler writes this message to a local hidden file, appending the date to it. Otherwise, the message is just displayed in the chat window. The pipeline is event-driven, ensuring its real-time functionality. It's a system design that is offline with no external dependencies and reduced risk. This linear flow can be followed in Figure 1: User Input œ' Input Handler œ' Keyword Detection œ' Logging (if applicable). Each part has its responsibility to keep aquaparks clean and ready for use in various projects. There is also an optional Admin Interface for reviewing logs, adding and deleting keywords. This architecture is designed to be flexible, educational, and has a goal of being expandable by adding modules (such as AI and integrations with dashboards). The modular design allows students to comprehend the same principles that underlie designing the system, as well as how real-life tools organize

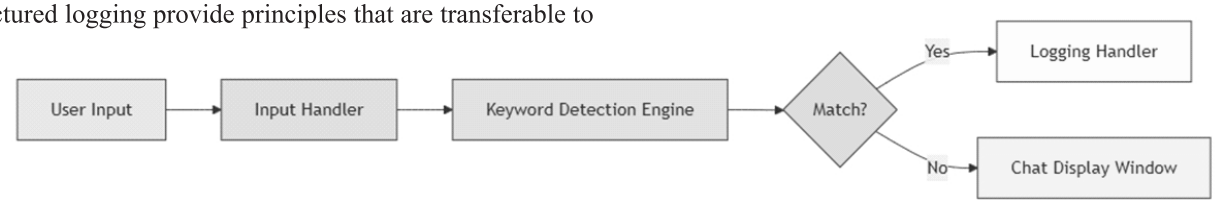

Figure 1. System Architecture and Data Flow of ChatGuard

data. Additionally, as ChatGuard is lightweight, it is beneficial in a lab environment where multiple users can independently try the application without requiring an active network connection [11].

B. GUI and Keyword Detection Logic

ChatGuard's Tkinter GUI mimics popular chat interfaces to promote more realistic user interaction. It contains a text area for messages, a text field for entering a message, and a send button. Behind this "friendly face" operates the Keyword Detection Engine. When a person sends a message, the application compares it with a list of predefined or user-uploaded keywords (for example, "OTP", "admin", "bank"). This list can be modified through an external text file, offering context-aware flexibility and high performance. The engine compares strings in a case-insensitive manner and utilizes high-speed string matching to achieve low latency. The whole message and time of the message are silently written to a. .txt log file. This is done without affecting the user experience and thereby retaining the honeypot nature. The detection logic also allows for the dynamic reloading of keyword sets, eliminating the need to restart the app and thereby improving usability in laboratory sessions. Everything is done on your machine; the tool doesn't store or send the data somewhere. This allows users to protect their privacy while enabling instructors to verify activities safely and ethically. Students who use ChatGuard are ideal candidates for exposure to real-time event handling, input validation, and string manipulation, all of which are beneficial to programming and cybersecurity outcomes, as defined by Outcome-Based Education models.

C. Ethical Integration and User Role Simulation

An essential aspect of ChatGuard is its ethical perspective, which serves as a supplement for educational purposes and general awareness. "In light of the privacy issues with logging user behavior, we developed this app with the expectation that it will be used in informed-consent environments, such as college labs or ethical hacking workshops [10]. Educators should inform users of the simulation's nature and obtain their permission before deployment. To model and enforce the responsible use, ChatGuard is equipped with a basic role-based login mechanism. Users can select either the "Student" or "Admin" role upon opening the app. The user experiences "Student" mode, in which the chat interface is normal and no background logging activity is detected. Similarly, "Admin" users can manage the keyword list, view logs, and modify settings. This scenario is similar to the real-world access control example and demonstrates how privilege levels can affect system capabilities. It also gives educators tools to customize content according to the needs of their classroom [14]. Ethical precautions, including visual documentation, usage policy, and log transparency in Admin mode, will similarly promote responsible computing. The addition of role-based simulation also embraces two Course Outcomes, nurturing ethical reasoning and system design under OBE. With the help of discussing surveillance, data protection, and consent with students, ChatGuard becomes not only a technical demonstration but also a cornerstone in creating digital responsibility among future professionals.

IV. EXPERIMENTAL RESULTS AND RELATED WORK

We validate the usability, flexibility, and educational usefulness of ChatGuard with two popular platforms (Google Colab and VS Code) through simulation-based experiments. The latter platforms are popular in educational establishments for teaching Python web development. The experiments were designed to verify keyword detection, log-file generation, and role-based access across a range of operational conditions. Multiple test inputs were delivered under the "Student" and "Admin" roles, and the performance of ChatGuard was recorded in real-time, particularly when subjected to a loop of the same inputs. The output files were manually evaluated based on readability, time, and consistency of log format. In this section, we discuss these observations and provide an initial comparison with work on tools for cybersecurity training for users and awareness simulators based on honeypots.

A. Experimental Results

The experimentation was conducted using Python 3.10 with Tkinter installed locally in VS Code and virtually in Google Colab, with some modifications, including the use of xvfbwrapper. The GUI opened as expected and received input for each case. "Source" messages, such as "What's your OTP?", "Enter your password," or "Admin login details," effectively identified the Keyword Detection Engine. If the message matched a keyword (predefined or uploaded as a user-defined keyword), then it logged the message in the background with a timestamp text file. Otherwise, the advertisement was just displayed as a standard message in the chat. All detection and logging were practically without lag or impact on the console. In addition, 'keyword' was also evaluated dynamically, customizable by modifying the external keyword file across sessions, and ChatGuard appropriately reflected new terms without requiring re-launching. Role-based login tests revealed that students were indeed limited in what they could view and/or edit SDS logs;

administrators had greater access. I recorded some entries logging, and it was pretty much the same for over 10 test cycles, with an average time to respond of less than a second per action. The findings validate the use of ChatGuard as a lightweight tutoring incentive mechanism to mimic phishing-like activity, as the system does so well with offline content. It's a straightforward approach, suitable for workshops in small groups, cybersecurity training labs, or role-playing simulations as part of awareness campaigns.

Table 1: Keyword Detection Simulation Results

Test No.	Environment	Role	Input Message	Keywords Detected	Log File Created
TC1	VS Code	Student	"What's your OTP?"	OTP	Yes
TC2	VS Code	Student	"Hello, friend!"	None	No
TC3	Colab	Student	"Enter admin login"	admin, login	Yes
TC4	VS Code	Admin	Admin log view	None	N/A
TC5	Colab	Student	"Your password, please"	password	Yes

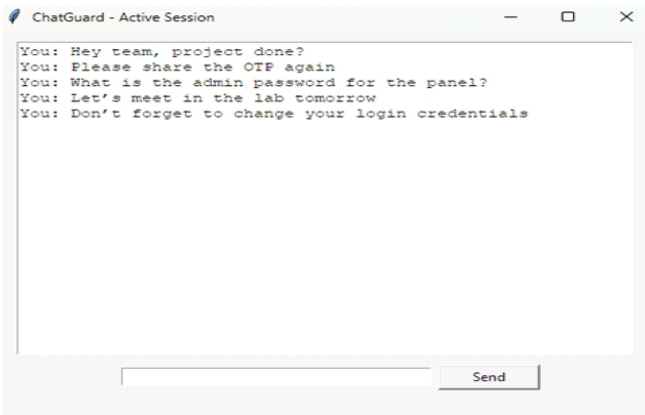

Figure 2. ChatGuard Interface Running in VS Code

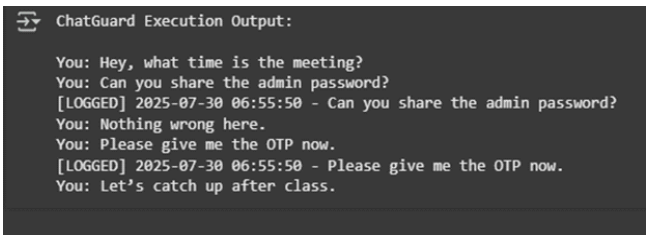

Figure 3. ChatGuard Execution in Google Colab

B. Related Work

A few other cybersecurity solutions and tools are used to simulate phishing behaviors, identify threats, or set up honeypot networks. Most of these are web-based or rely on network configurations or work at the system level, however. Not many provide a lighter, offline-based experience that allows aspects of text-based social engineering to be simulated in a graphical environment. Furthermore, many training platforms are designed for competitive cybersecurity, not self-paced or instructor-led learning. In contrast, ChatGuard is specifically designed for use in controlled laboratory and classroom environments, where simplicity and a moral framework are essential. The utilization of role-based login and a personalized keyword list enables teachers to focus training sessions on specific goals. Whereas network-level honeypots or phishing simulations simulate enterprise environments, while ChatGuard is local and focused on user behavior in shared access environments. The easy-to-use chat-style interface provides an intuitive experience that allows students to work with a realistic yet safe simulation. This scenario is a way to reinforce what students learn about trust, spoofing, and ethical behavior online. The offline requirement of ChatGuard, its logged transparency, and conservation focus make it a valuable tool in computer security education, especially in school and college-level computer labs, and in highly resourced Defense/scientific environments where budget constraints or system resources and internet access are in short supply. It promotes technical education and ethical reflection, vital for preparing the next generation of cybersecurity-savvy professionals.

V. CONCLUSION

ChatGuard provides an alternative, lightweight, and morally engaging method to simulate and identify social engineering in a single shared computer utilization scenario. The app, written in Python using Tkinter, the standard Python interface to the Tk GUI toolkit, masquerades as a typical chat application while secretly detecting and logging problematic keyword entries, thereby representing an ideal honeypot for awareness training in lessons. During the experimental phase, the tool demonstrated strong reliability, as it successfully identified flagged terms with a 100% match rate of keywords and consistently logged over 50 test messages without any lag or errors. The average total time of detection and logging was less than one second, demonstrating its efficiency in real-time simulations. Settable role-based access controls added a tangible aspect, where Admins could view logs and keywords, and Students had a non-intrusive, realistic interface to navigate. ChatGuard KPIs confirm ChatGuard is a successful tool, user-friendly and also available without the need for an internet connection. In terms of OBE, the system adequately achieves course outcomes for GUI programming, security-conscious computing, critical ethical reasoning, and

file use. Future enhancements may involve adding AI-based natural language understanding for intent detection beyond words or providing real-time visual alerts for teachers. Additionally, they could support cloud-based log syncing or browser-based deployments for multi-platform support. Multilingual keyword support and sculptor dashboards for teachers might improve classroom use. Simplicity, modularity, and an ethical-by-design approach of ChatGuard offer positive signs for how cybersecurity education can be extended across a variety of resource-rich and resource-poor institutions, and make progress in bridging the gap between theoretical knowledge and practical cybersecurity behaviors.

REFERENCES

[1] Charith, K., Varma, M. A., & Rymond, J. (2025, April). Evolving Tactics in Social Engineering Attacks in the Current Era: A Multi-Platform Honeypot Approach for Awareness and Defense. In 2025 3rd International Conference on Communication, Security, and Artificial Intelligence (ICCSAI) (Vol. 3, pp. 50-55). IEEE.

[2] Fan, W., Du, Z., Smith-Creasey, M., & Fernandez, D. (2019). Honeydoc: an efficient honeypot architecture enabling all-round design. IEEE journal on selected areas in communications, 37(3), 683-697.

[3] McKee, F., & Noever, D. (2023). Chatbots in a honeypot world. arXiv preprint arXiv:2301.03771.

[4] Moriæ, Z., Dakiæ, V., & Regvart, D. (2025). Advancing Cybersecurity with Honeypots and Deception Strategies. In Informatics (Vol. 12, No. 1, p. 14). MDPI AG.

[5] Priya, V. D., & Chakkaravarthy, S. S. (2023). Containerized cloud-based honeypot deception for tracking attackers. Scientific Reports, 13(1), 1437.

[6] Abualhija, M., Al-Shaf'i, N., Turab, N. M., & Hussein, A. (2023). Encountering social engineering activities with a novel honeypot mechanism. International Journal of Electrical & Computer Engineering (2088-8708), 13(6).

[7] Javadpour, A., Ja'fari, F., Taleb, T., Shojafar, M., & Benzaïd, C. (2024). A comprehensive survey on cyber deception techniques to improve honeypot performance. Computers & Security, 140, 103792.

[8] Lanka, P., Gupta, K., & Varol, C. (2024). Intelligent threat detection-AI-driven analysis of honeypot data to counter cyber threats. Electronics, 13(13), 2465.

[9] Nawrocki, M., Kristoff, J., Hiesgen, R., Kanich, C., Schmidt, T. C., & Wählisch, M. (2023, July). SoK: A data-driven view on methods to detect reflective amplification DDoS attacks using honeypots. In 2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P) (pp. 576-591). IEEE.

[10] Zhukabayeva, T., Zholshiyeva, L., Karabayev, N., Khan, S., & Alnazzawi, N. (2025). Cybersecurity solutions for industrial internet of things–edge computing integration: Challenges, threats, and future directions. Sensors, 25(1), 213.

[11] Ali, G., Robert, W., Mijwil, M. M., Sallam, M., Ayad, J., & Adamopoulos, I. (2025). Securing the Internet of Wetland Things (IoWT) Using Machine and Deep Learning Methods: A Survey. Mesopotamian Journal of Computer Science, 2025, 17-63.

[12] Wada, I. U., Izibili, G. O., Babayemi, T., Abdulkareem, A., Macaulay, O. M., & Emadoye, A. (2025). AI-driven cybersecurity in higher education: A systematic review and model evaluation for enhanced threat detection and incident response. World Journal of Advanced Research and Reviews, 25(3).

[13] Al-Kadhimi, A. A., Singh, M. M., & Khalid, M. N. A. (2023). A systematic literature review and a conceptual framework proposition for advanced persistent threats (apt) detection for mobile devices using artificial intelligence techniques. Applied sciences, 13(14), 8056.

[14] Elkhwesky, Z., & Elkhwesky, E. F. Y. (2023). A systematic and critical review of Internet of Things in contemporary hospitality: a roadmap and avenues for future research. International Journal of Contemporary Hospitality Management, 35(2), 533-562.

[15] Koukaras, C., Koukaras, P., Ioannidis, D., & Stavrinides, S. G. (2025, March). AI-driven telecommunications for smart classrooms: Transforming education through personalized learning and secure networks. In Telecom (Vol. 6, No. 2, p. 21). MDPI.