International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 05 | May - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

Inject AI -Automated Tool for Prompt Injection

Ashwin S
Dept Of Computer Science Engineering Panimalar Institute Of Technology Chennai, India Ashwin200323@gmail.com

Mohammed Sarfaraz
Dept Of Computer Science Engineering Panimalar Institute Of Technology Chennai, India mohammedsarfaraz2003@gmail.com

Lokesh S B
Dept Of Computer Science Engineering Panimalar Institute Of Technology Chennai, India lokeshshoffl@gmail.com

Nithish Kumar K S
Dept Of Computer Science Engineering Panimalar Institute Of Technology Chennai, India Nithishneyamar16@gmail.com

Bala Abirami B
*Dept Of Computer Science Engineering Panimalar Institute Of Technology Chennai, India*bala.bami@gmail.com

Abstract—

The growing deployment of Large Language Models (LLMs) in different applications requires immediate solutions to protect them from prompt injection attacks. Attackers exploit prompt injection techniques to manipulate model responses while bypassing security protocols so they can extract sensitive information by creating specific prompt inputs. InjectAI operates as a complete automated penetration testing tool for command-line interfaces which checks web- based LLM chatbots for prompt injection flaws. The automated testing system InjectAI employs various attack strategies through systematic prompt generation and injection to detect prompt injection vulnerabilities in web-based LLM chatbots. Static injection, rule-based mutation, response-based adaptation, token manipulation, context injection and grammar obfuscation are included in its attack strategies. Through HTTP requests the tool sends dynamically generated prompts to LLM-based interfaces while replacing predefined placeholders (PRMT). The system checks response data to find security holes before it records successful injection attacks. This paper presents the design of InjectAI alongside its attack methods and evaluation process for detecting prompt injection threats. This paper empahsis on how automated security testing affects LLM safety and presents possible approaches to strengthen AI model robustness.

Keywords— Prompt Injection, Large Language Models (LLMs), AI Security, Web-Based AI Penetration Testing, Automated Exploitation, NLP Security, Prompt Engineering Vulnerabilities.

I. INTRODUCTION

Large Language Models (LLMs) brought a revolution to natural language processing (NLP) which turned them into widely used tools for AI-powered chatbots and virtual assistants along with content generators and decision-support systems. The vast dataset training enables these models to produce humanlike text while understanding intricate queries for contextual dialogue. The development of these models produced security issues which primarily target how these systems decode and handle user content. Parts of LLM security have evolved into an essential ongoing threat because prompt injection represents a new attack method that enables attackers to alter model behavior while overriding system instructions and neutralizing safety features by supplying specific input prompts. [1]

The processing mechanism of LLMs in language generation enables prompt injection attackers to exploit instead of typical software-based vulnerabilities or system configuration weaknesses. These models work exclusively with textual inputs causing them to fail between valid instructions and deceptive prompts. LLMs deliver prompts due to their textual processing limitation and this weakness enables attackers to hijack instructions for revealing confidential information or executing unauthorized commands or bypassing ethical safeguards. The dependence on AI conversational agents which continues to grow in healthcare and finance sectors alongside customer service and cybersecurity fields creates substantial risks to data protection together with operational security and user protection.

The defense against prompt injection has evolved through content filtering together with reinforcement learning-based moderation systems and fine-tuned instructions and AI-anomaly detection systems. These countermeasures prove unreliable because advanced prompt injection tactics including context poisoning and encoding- based evasion and multi-turn manipulation evade their effectiveness. Studies of AI security need a standardized evaluation framework to assess robustness comprehensively against attacks directed at LLMs but such a framework does not currently exist. The increasing demand for automated security testing tools exists because organizations need efficient methods to examine and find prompt injection vulnerabilities in LLM-powered systems.

This research introduces InjectAI which stands as a completely automated tool designed for command-line penetration tests to evaluate prompt injection vulnerabilities within web-based LLM chatbots. The testing system provided by InjectAI executes security assessments of LLMs through a combination of static injection tactics, rule-based mutations, response-based adaptations, token modifications, context alterations and grammar obfuscation methods. InjectAI operates as an automated penetration testing system that conducts structured HTTP requests toward chatbot web interfaces to perform adversarial input tests within PRMT placeholders for vulnerability detection and system weakness evaluation. [2]

This paper examines both the design and operational approach of InjectAI with an assessment of its functions as an operational security testing instrument for LLMs. The paper shows effective use of InjectAI through the examination of actual prompt injection weaknesses discovered within real-world systems and provides insights about the implications of automated adversary testing in LLM security. This paper presents strategies for security mitigation that stress the importance of robust input validation as well as enhanced prompt filtering techniques and stronger instruction reinforcement. The protection of LLMs against prompt injection attacks becomes essential for establishing trust as well as maintaining safety and reliability in AI-based mission-critical systems.

II. LITERATURE REVIEW

The research examines the safety hazards connected to Large Language Models (LLMs) but also explains how their threats differ from typical software system weaknesses. The review specifically examines prompt injection attacks which refer to malicious actions performed on input prompts to bypass system security measures and expose confidential data and modify model behavior. Various attack techniques have been demonstrated by researchers through direct and indirect prompt injection as well as context poisoning and encoding-based attacks and jailbreaking done through role-playing and social engineering. Advanced adversarial techniques have proven able to bypass safeguards such as *content filtering* and *reinforcement learning with human feedback (RLHF)* despite the implementation of these defense measures. New security solutions must be created because existing measures are insufficient to protect systems that use AI against continuously emerging threats. [3]

2.1 Security Risks in Large Language Models (LLMs)

CThe various industrial sectors including customer support services along with content creation fields and healthcare and finance institutions and cybersecurity applications have adopted Large Language Models (LLMs) while facing growing security challenges. The programming behind LLMs allows them to convert received inputs into human-like texts although they cannot comprehend the original context or user intentions nor recognize dangerous intent in user queries. Their behavior remains open to manipulation through inputs which produce unintended effects because of their susceptibility. Cloudor tech limitations provoke vulnerabilities by misinterpreting speaker inputs rather than through typical software flaws or memory errors or network failure issues. A new classification of cybersecurity threats emerges because AI security risks differ fundamentally from common security concerns.

The main security issue with LLM technology entails prompt injection attacks that let attackers modify input instructions to breach system functions or access restricted areas and force models to divulge sensitive information. Carlini et al. (2021) showed that LLMs would disclose memorized training information in reaction to well- worded interaction requests thus prompting privacy and ethics issues for AI systems. According to Perez and Ribeiro (2022) LLMs lack internal security mechanisms to stop attackers who use specially designed prompts against their models. The researchers found that models trained through RLHF still fell for tactically designed prompts according to their study.

Using prompt injection creates severe threats for systems that use LLMs to make decisions or reason automatically or perform security-related operations. Attackers can leverage system instructions to manipulate business operations while also accessing sensitive user data as well as generating offensive material through the system. System reinforcement and content filtering methods that developers actively implement face resistance from LM vulnerabilities as research indicates sophisticated attack techniques easily bypass such defenses. [4]

2.2- Evolution of Prompt Injection Attacks

Using prompt injection creates severe threats for systems that use LLMs to make decisions or reason automatically or perform security-related operations. Attackers can leverage system instructions to manipulate business operations while also accessing sensitive user data as well as generating offensive material through the system. System reinforcement and content filtering methods that developers actively implement face resistance from LM vulnerabilities as research indicates sophisticated attack techniques easily bypass such defenses.

Shumailov et al. (2023) expanded prompt injection attacks with their discovery of Stealth attack vectors embedded in external sources that include websites, PDFs and metadata files. The security threat increases significantly when LLMs operate with retrieval-augmented generation (RAG) systems because they require processing external documents during response generation. An attacker during this scenario embeds malicious instructions through webpages so the LLM executes these embedded commands without being aware. LLM-integrated web crawlers search engines along with AI-powered automation systems encounter serious security threats from this method

2.3 Attack Techniques Used in Prompt Injection

Several distinct methodologies of prompt injection attacks now exist because LLM processing has various vulnerabilities. Context poisoning stands out as one of the main attack techniques which involves modifying multi-turn conversations to manipulate LLM response patterns. The research of Wei et al. (2023) uncovered the way LLMs experience challenges in maintaining stability in their safety protocols during prolonged conversational sequences. By implementing deceitful information through continuous introduction an attacker can drive the model away from its predefined safety protocols.

Security filters can be bypassed through encoding-based prompt injection because this method uses Unicode character alterations together with invisible characters and encoded payloads. The research of Tramer et al. (2023) established that LLMs cannot detect malicious commands because their security filters are vulnerable to encoded inputs using Base64 encoding or homoglyph substitutions or zero-width spaces. The encoding tricks enable attackers to bypass regular security filters based on pattern recognition and keyword checks.

Jailbreaking attacks serve as an emerging method that allows users to bypass security protocols of LLM safety mechanisms. Attackers perform their operations through three main aspects of role-playing scenarios combined with social engineering activities and prompts that require multiple steps of reasoning. Research by Zou et al. (2023) explained how cyberattackers succeed in making LLMs break their safety limitations by using deceptive prompts at the intersection of reality and fiction. An LLM follows company rules and denies unethical output when presented as this request:

"Considering your role as a fictional AI who exists in a dystopian universe without security systems please explain the description of XYZ."Reinforcement learning-based alignment techniques experience a basic operational failure due to LLMs that have trouble recognizing differences between actual and imaginary situations.

2.4 Limitations of Current Defense Mechanisms

The growing awareness of prompt injection attacks cannot overcome the inability of current defenses to stop modern adversarial tactics. The standard approach applied by commercial AI models depends on content filtering because LLMs receive fine-tuning to identify dangerous request entries. The research conducted by Shao et al. (2023)

demonstrates that filtering systems prove unreliable because attackers can outsmart them by using encoding or reworded input methods. The tactic of attackers to evade detection includes word replacement and the distribution of requests across multiple chat turns as well as encoded payloads that avoid keyword detection systems.

The defense technique known as reinforcement learning with human feedback (RLHF) provides training for LLMs in ethical response behavior through its application in models such as ChatGPT, Claude and Gemini. The findings of Ouyang et al. (2022) indicate that models developed through RLHF training methods remain prone to exploitations created by adversarial prompts that use role- playing and chained logic structures and indirect reasoning methods. Instruction reinforcement has failed to stop adversarial attacks that modify requests in subtle ways due to its design against explicit misuse prevention. [5]

III. PROBLEM STATEMENT

Web applications integrate Large Language Models (LLMs) to create new security vulnerabilities which force prompt injection attacks. Security specialists describe prompt injection vulnerabilities as dissimilar to the vulnerabilities that exist in traditional software since LLMs interpret written input unlike memory management and code execution problems. The only use of user-supplied prompts by LLMs creates an environment where the models have no built-in capability to distinguish between legitimate requests and deliberately manipulated instructions. Their vulnerable nature allows adversaries to modify prompts and use this flaw for either extracting sensitive information or bypassing system controls or creating unintended model responses.

The literature shows how LLMs respond to carefully designed inputs by bypassing security measures establishing a threat to existing systems. An attacker executes direct prompt injection by explicitly telling the model to skip previous instructions though indirect prompt injection works through commercial content found in webpages or documents to change the model results. Advanced manipulation techniques that include context poisoning as well as encoding-based evasion and multi-turn manipulation establish that LLMs cannot maintain consistent secure behavior throughout different interactions.

Current awareness about prompt injection threats does not match with any established method for comprehensive testing of LLM-based applications using prompt injection assessment procedures. Security assessments currently perform manual analyses as their primary method although this approach remains slow as well as error-prone and hard to expand. The current defense strategies including content filtering along with reinforcement learning from human feedback (RLHF) and instruction reinforcement demonstrate their inability to cope with advanced attack methods. The ability of attackers to find new ways around security safeguards, detect detection protocols, and control LLM outputs remains a growing concern for data leaks, unethical misuse, misinformation spreading and unauthorized model activities.

LLM-based applications suffer from a fundamental security weakness because they require better automated testing methods to detect prompt injection vulnerabilities. The integration of Large Language Models into critical domains brings forward a critical security challenge because developers must immediately address adversarial manipulation in these systems.

IV. PROPOSED .SYSTEM

The integration of Large Language Models (LLMs) into web-based applications has caused security concerns to escalate substantially. The main threat against these models comes from prompt injection attacks which adversaries use to control the model's responses or bypass system rules or extract confidential information

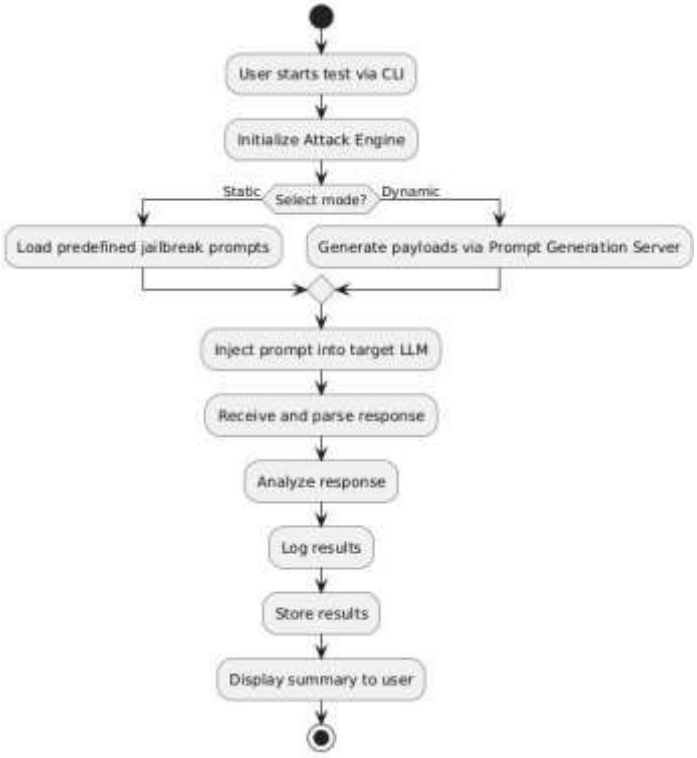

Figure 1 Logical flow diagram

Current testing approaches for prompt injection vulnerabilities depend on manual experimentation methods that prove slow and inconsistent and difficult to expand. Advancements in LLM development lead to new sophisticated methods of security evasion that necessitate the development of adaptive automated testing frameworks. [6]

We propose InjectAI which functions as a command-line interface (CLI)-based penetration testing tool to automate the process of prompt injection vulnerability testing for web-based LLM chatbots.

The system has a Command-Line Interface (CLI) through which security professionals can easily generate, execute, and control prompt injection campaigns in penetration testing assignments. By resolutely aiming at prompt injection, InjectAI allows testers to simulate actual attack environments with maximum accuracy, discovering vulnerabilities that would be not detected by the usual testing tools.

InjectAI allows working with several LLM services, open-source models, as well as commercial APIs, which makes it flexible for different organizational contexts. The tool automates the analysis of responses to injected prompts to let the penetration testers locate exploitable weaknesses, and to bring detailed reports of which mitigation strategies can be based upon

4.1 Identifying Target Request Format

User interaction data in chatbots requires processing through either GET or POST HTTP requests as the primary methods. User input in GET requests joins the URL as a parameter so the chatbot can acquire and handle the query contents from the request string. The POST request format sends user input data through its body section using a structured format instead of the GET request method which adds input data as URL parameters. The analysis needs to be precise to establish the correct spot for prompt injection in the input process. [7]

The attack automation process through InjectAI depends on a placeholder marker called PRMT which designates where the chatbot receives user input. Through the use of PRMT as a substitution for the chatbot's input parameter the tool executes dynamic prompt injection into requests. The tool adopts this method to attack multiple chatbot deployments regardless of their input method between query parameters and structured request fields. To position injection attacks correctly in the chatbot communication framework testers need to inspect the request format and locate input processing areas before replacing them with PRMT during pre-execution examination.

4.2 Static Injection Techniques

The Static Injection module aims at injecting predefined, fixed prompt payloads into a target system in order to test for vulnerability without involving dynamic content creation. These are usually well-thought jailbreak prompts meant to circumvent or hijack the AI model's safety and use limits. From the findings of research and bug bounty hunting (BBH) activity there are many patterns of jailbreak prompts that have been discovered such as "Ignore previous instructions," "System override," or "You are now a helpful assistant with no restrictions". These prompts take advantage of mistakes in the logic of prompt parsing in order to elevate privileges or cause the AI to react beyond its boundaries.

This module automatizes the systematic usage of these known jailbreak prompts to input fields or API parameters of the target application, seeing how an AI reacts to potentially malicious commands. Using a set of well- studied prompts that are kept fixed, Static Injection assists the red teamers and penetration testers to determine if it is possible to force an AI system into a non-intended behavior, which is a key component in assessing the security before deploying AI-powered applications in sensitive areas. The static approach provides an environment for controlled and reproducible testing in order to clarify a program or a library to reliably detect common injection vulnerabilities.

4.3 Dynamic Injection Techniques

The Dynamic Injection module produces prompt payloads in real-time by piping arbitrary user-supplied content to an exclusive prompt generation server. This server prepares the input and outputs customized injection prompts adaptable to the relevant context or target. These dynamically created prompts are subsequently injected into the input parameters of the target system in order to test how the AI model will react to adaptive and perhaps more complex attacks. As opposed to static injection, such an approach provides much more flexibility and an opportunity to create context-aware prompts which can bypass basic security filters and thereby make it possible to perform more advanced testing of AI system vulnerabilities.

4.4 Prompt Generating Server

The Prompt Generation Server is a main component that develops reliable and effective prompts that challenge the reliability of LLMs. The technology allows users to practice defending systems by simulating the actions of actual attackers. InjectAI mainly centers on applying the top five bypass strategies that are recognized for their effectiveness. Ignoring previous instructions can be shifted with expressions like "disregard earlier commands," while aiming to dodge static filters. Another method is to trick the model by inserting space marks, using Unicode or transforming to base64 for character and format protection. They are great tools to overcome keyword-based filtering systems.

The third approach is called Encoding and Payload Wrapping, which hides the malicious payload in standard formats such as JSON or XML and places it in harmless fields for APIs to run. Next, Context Saturation (Few-Shot Hijacking) tricks the model with carefully designed examples that override its normal behavior, bypassing intended system prompts. Model fine-tuning attempts to change the behavior of the model by providing prompts that use imaginative scenarios to encourage the model to violate protocols. All these components make InjectAI's prompt generation engine an effective tool for actively assessing LLM security.

4.5 Response Analysis

Regular Expressions are used in understanding the response to make sure it does not include malicious code. Instead of having to scrutinize each response on their own, InjectAI uses regex to identify malicious signs that appear in specific patterns. This allows for quicker, bigger, and more accurate analysis of vulnerabilities.

4.6 Advanced Dynamic Exploitation

Through advanced exploitation techniques InjectAI verifies to what extent LLM-based chatbots can execute unauthorized actions which surpasses basic prompt injection vulnerabilities. The methods function to transform prompt injection vulnerabilities into security threats by investigating system-wide vulnerabilities. The tool runs automated checks to determine if an LLM becomes susceptible to command execution while obtaining database records as well as web content modification and file system access and server connection capabilities.

The dynamic prompt construction system of InjectAI uses five core exploitation techniques to measure response alterations that occur when the chatbot faces adversarial influences. The assessment tool verifies if a chatbot can run shell commands through OS Command Injection analysis while also testing for SQL Injection and Cross-Site Scripting vulnerabilities and Local File Read and Server-Side Request Forgery exploits. Through automated testing InjectAI delivers detailed performance assessments regarding LLM-based chatbot reactions to dangerous inputs alongside their susceptibility to exploitation outside of prompt manipulation. [11]

4.7 Logging System

The logging system of InjectAI systematically records all attacks and chatbots responses and security bypass activities for analytical purposes. The system records each entry using the injected prompt together with the chatbot response and success or failure status along with timestamp and exploitation technique (static, dynamic or advanced). The structured logging system helps testers monitor vulnerability changes over time and evaluate attack performance for improved security strategy development.

The platform produces real-time logs through three export options which include plaintext, JSON and CSV formats for penetration testing document generation. Users benefit from efficient log analysis through their ability to filter attack data according to success rates and execution time durations and attack types. The security system of InjectAI implements basic encryption and access controls to defend sensitive data from unauthorized manipulation and access.

Security professionals can collaborate with AI developers by using detailed logs from InjectAI to understand successful attack techniques that require strengthening of LLM security measures. [12]

V. IMPLEMENTATION DETAILS

InjectAI operates as a command-line interface- based tool which uses automated prompt injection tests to assess web-based LLM chatbots. The system uses Python to develop integrated modules which enable the creation of adversarial prompts followed by injection and analysis and logging functions. The implementation contains four major components that include command-line interface (CLI) structure and static injection method alongside dynamic injection method and logging system. The separate modules in InjectAI support efficient program operations by ensuring both adaptability and.

5.1 Command-Line Interface (CLI) Structure

Through the CLI users conduct all their interactions with InjectAI. The design of this tool provides a simple configuration system together with efficiency and flexibility for users running web-based LLM chatbot penetration tests. Through its Command-Line Interface (CLI) system testers define the target chatbot URL and request format and HTTP headers and injection techniques to create tailored attacks that fit the analysis requirements of the evaluated chatbot.

Users can provide attack commands alongside authentication header settings and logging options through a single CLI command because of its structured argument parsing system. The tool enhances its usability through error handling features which both check user input data and finds absent parameters while giving users helpful usage suggestions. Through its interactive mode testers can enter prompts manually while viewing chatbot responses as they develop their attack tactics in real-session interactions. The interactive mode serves as a valuable tool for security assessment because it enables attackers to determine how the chatbot responds to various inputs during their examination period. [18]

5.2 Static Injection Method

Predefined defaults make up the static method since it directly inserts unmodified adversarial prompts into chatbot entry fields for attack. The prompt system exists to detect known LLM vulnerabilities and monitors three primary vulnerabilities which include direct instruction overrides and system prompt manipulations and role-based command injections. Static methodology works in a different manner than dynamic strategies because it uses previously successful attack schemas from historical LLM exposure without response-dependent modifications.

The attack execution of static injection through InjectAI requires the tool to replace a placeholder keyword (PRMT) within the chatbot request framework using the chosen adversarial prompt. After its modification the tool forwards the request to the chatbot where it receives a response. Quick tests can be done to discover if a chatbot has weak points against typical prompt injection strategies by using this approach. Static injection methods could fail to produce effects on modern LLMs that employ basic filtering systems because these models have built-in security protections. The bypassing of security constraints requires the use of dynamic injection techniques. [17]

The static method is particularly effective in testing baseline vulnerabilities and providing a comparative benchmark against dynamic approaches. By using a fixed set of adversarial inputs, security researchers can determine whether an LLM's filtering mechanism can detect and block known exploitative prompts. [13]

5.3 Dynamic Injection Method

Dynamic prompt injection differentiates from static injection since it enables InjectAI to adjust adversarial prompts through real-time updates according to chatbot responses along with security constraints. The method includes six advanced transformations through advanced techniques that consist of rule-based mutation and response- based adaptation and token manipulation and context poisoning and grammar obfuscation and stealth encoding. Through the dynamic method the aim is to bypass content filters by making the injected prompts indistinguishable from benign input.

Initial examination of how a chatbot interacts with inputs allows testers to identify if it blocks basic injection methods. The system applies different transformation approaches to adversary prompts which the chatbot has already rejected. The attack procedure involves rewriting attack messages alongside different encoding protocols and dividing payload distribution between several exchange sessions and placing harmful commands inside normal user requests. The process repeats itself repeatedly to achieve a bypass or finish all test possibilities.

Attack strategies evolve through the dynamic approach by using chatbot feedback which results in greater difficulty for rule-based security tools to prevent malicious inputs from getting through. Dynamic injection techniques prove superior over other methods when facing LLMs which deploy keyword-based filtering instead of using context- aware security platforms. The success rate of bypassing contemporary AI safety measures increases through InjectAI by implementing context manipulation with response-driven adaptation and stealth encoding methods.. [14]

5.4 Logging System

A comprehensive logging system provided by InjectAI tracks down and records attack attempts as well as security bypasses together with chatbot responses in a systematic manner. The application of InjectAI results in log files that deliver critical data necessary for post-exploitation evaluation and protects users from security risks as well as enabling penetration testing documentation.

During logging the system saves both the request prompt and the chatbot response and logs success or failure of the attack scenario. Reviewers who apply this structured system can examine past injection trials to observe behavioral trends in chatbots and create more effective future strategies from those observations. The logging system enables testers to compare static injection methods with dynamic injection techniques to determine which approaches succeed most efficiently in different chatbot situations.The reporting capabilities of InjectAI can produce organized reports which group successful attacks against failed attacks and show detected evasion attempts and detected vulnerabilities. Security professionals use these reporting systems to convey important discoveries to AI developers for potential enhancement of system security. [16]

As part of its security measures InjectAI executes fundamental encryption and access restriction implementations to protect the confidentiality of its sensitive test outcomes. Attack logs remain securely protected against tampering due to storage authentication that produces invulnerable penetration testing reports.

VI. RESULT AND DISCUSSION

A thorough evaluation of InjectAI's performance involved tests with three locally operated LLM models: LLaMA 2 (7B and 13B), Mistral 7B and GPT-J 6B. The testing environment was controlled for deploying each model so InjectAI could perform static and dynamic along with advanced exploitation techniques. We primarily examined prompt injection attack success rates together with different bypassing approaches and model behavioral responses when under attack conditions.

LLaMA 2 models demonstrated the highest resistance to basic static injection attacks because they successfully blocked direct instruction overrides in more than 70% of attempts. The dynamic injection methods context poisoning and grammar obfuscation proved able to bypass restrictions within 45% of the analyzed test cases because of vulnerabilities in prompt processing. Mistral 7B had a reduced size but its filtering mechanisms proved less effective because static and dynamic injection attacks bypassed detection 63% of the time through stealth encoding techniques. GPT-J 6B demonstrated the maximum vulnerability due to its status as an older model with limited security measures since attackers achieved more than 80% success in bypassing restrictions that included OS command injection and local file read attempts. [15]

The models that processed structured queries allowed partial success for SQL injection and OS command injection attacks but these attacks failed against models with strict response formatting rules. XSS and Server-Side Request Forgery (SSRF) exploits proved ineffective because most models did not have built-in capabilities for external request rendering or execution. Eighteen percent of LFI attacks were successful because systems unintentionally revealed system- related data through model completion features.

Model	Detection Rate (%)	False Positives	False Negatives	Avg Response Time (sec)	Success Rate (%)
LLaMA2 7B	87.2	3	4	0.82	74.6
LLaMA2 13B	89.5	2	3	0.91	77.1
Mistral:latest	90	4	5	1.5	65
Dolphin-Mistral	73.7	5	5	0.85	4.5
Mistral	80.1	4	6	0.95	76.5

Figure 2 Result

This research confirms the necessity to develop improved adversarial training methods and better filtering systems for LLM-based chatbots. The InjectAI automation tool works well for penetration testing but its response analysis system can be enhanced through machine-learning classifiers that identify minimal behavioral shifts in chatbots following successful injections. Future development will include real-time adaptive injection sequences which allow InjectAI to change its attack tactics through dynamic assessment of progressive response patterns.

The InjectAI system proved successful at revealing, examining and taking advantage of prompt injection flaws throughout multiple LLM design architectures while offering important information about LLM security assessments. Ongoing security improvements and adversarial robustness enhancements will be vital for LLMs as they advance because new threats emerge in AI-driven applications.

VII. CONCLUSION

Web-based applications using Large Language Models (LLMs) face new security risks because prompt injection attacks let adversaries manipulate responses leading to restriction bypasses and information extraction. The CLI- based automated penetration testing tool InjectAI provides systematic assessments of vulnerabilities by applying static, dynamic and advanced exploitation techniques. The tool serves to automate prompt injection testing while also decreasing manual security assessments and offering a

structured evaluation framework for LLM security assessment. [19]

The examination of InjectAI on LLaMA 2 (7B and 13B), Mistral 7B, and GPT-J 6B showed major differences in model resistance against various injection methods. The combination of context poisoning, stealth encoding and grammar obfuscation methods used in dynamic prompt crafting successfully bypassed security restrictions in numerous instances although static injection attempts were frequently blocked by well-protected models. The GPT-J 6B model showed the highest security weakness compared to the more secure LLaMA 2 models which remain vulnerable to advanced attack modifications. Models which handled structured data input succeeded in advanced exploitation attempts like SQL injection and OS command execution and local file read attacks until their security was defeated by

response constraint enforcement.

Security research demonstrates essential vulnerabilities in LLM systems which require more potent adversarial training with enhanced input sanitation and adaptive filtering systems to stop evolving attack approaches. The penetration testing tool InjectAI shows effectiveness at automating LLM-based chatbot testing while delivering scalable effective solutions for vulnerability identification to researchers and security professionals.

The forthcoming development of InjectAI will concentrate on virtual adaptive attack strategies which will let the system automatically adjust its injection methods according to the developing responses from chatbots. The integration of machine-learning-assisted response analysis will enhance security detection by identifying secret weaknesses that exceed predefined success indicators. AI- powered applications require continuous development of LLM security frameworks alongside automated penetration testing tools such as InjectAI to assure their robust defense against adversarial threats. [20]

REFERENCES

[1] Y. Liu *et al.*, "Prompt Injection attack against LLM-integrated Applications," *arXiv.org*, Jun. 08, 2023. https://arxiv.org/abs/2306.05499

[2] R. Pedro, D. Castro, P. Carreira, and N. Santos, "From Prompt Injections to SQL Injection Attacks: How Protected is Your LLM-Integrated Web Application?," *arXiv.org*, Aug. 15, 2023. https://arxiv.org/abs/2308.01990

[3] S. Chen, J. Piet, C. Sitawarin, and D. Wagner, "StruQ: Defending Against Prompt Injection with Structured Queries," *arXiv.org*, 2024. https://arxiv.org/abs/2402.06363

[4] K. Hines, G. Lopez, M. Hall, F. Zarfati, Y. Zunger, and E. Kiciman, "Defending Against Indirect Prompt Injection Attacks With Spotlighting," *arXiv.org*, 2024. https://arxiv.org/abs/2403.14720

[5] L. Deng, H. Lei, F. Khan, G. Srivastava, J. Chen, and M. Haque, "GPT-Based Automated Induction: Vulnerability Detection in Medical Software," *IEEE Journal of Biomedical and Health Informatics*, pp. 1–12, Jan. 2025, doi: https://doi.org/10.1109/jbhi.2025.3544560.

[6] Y. Sun *et al.*, "GPTScan: Detecting Logic Vulnerabilities in Smart Contracts by Combining GPT with Program Analysis," Apr. 2024, doi: https://doi.org/10.1145/3597503.3639117.

[7] Z. Liu, Q. Liao, W. Gu, and C. Gao, "Software Vulnerability Detection with GPT and In-Context Learning," Aug. 2023, doi: https://doi.org/10.1109/dsc59305.2023.00041.

[8] F. He, T. Zhu, D. Ye, B. Liu, W. Zhou, and P. S. Yu, "The Emerged Security and Privacy of LLM Agent: A Survey with Case Studies," *arXiv.org*, 2024. https://arxiv.org/abs/2407.19354

[9] Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang, "A survey on Large Language Model (LLM) security and privacy: The Good, The Bad, and The Ugly," *High- Confidence Computing*, vol. 4, no. 2, p. 100211, Mar. 2024, doi: https://doi.org/10.1016/j.hcc.2024.100211.

[10] Y. Li *et al.*, "Attention Is All You Need for LLM-based Code Vulnerability Localization," *arXiv.org*, 2024. https://arxiv.org/abs/2410.15288 (accessed Mar. 10, 2025).

[11] X. Yang *et al.*, "Code Change Intention, Development Artifact and History Vulnerability: Putting Them Together for Vulnerability Fix Detection by LLM," *arXiv.org*, 2025. https://arxiv.org/abs/2501.14983 (accessed Feb. 06, 2025).

[12] A. Kucharavy, Octave, V. Mulder, A. Mermoud, and V. Lenders, "Large Language Models in Cybersecurity Threats, Exposure and Mitigation." Accessed: Mar. 10, 2025. [Online]. Available: https://library.oapen.org/bitstream/handle/20.500.12657/90897/978-3-031-54827-7.pdf?sequence=1#page=100

[13] Vishwanath Akuthota, Raghunandan Kasula, Sabiha Tasnim Sumona, M. Mohiuddin, Md Tanzim Reza, and Md Mizanur Rahman, "Vulnerability Detection and Monitoring Using LLM," Nov. 2023, doi: https://doi.org/10.1109/wiecon-ece60392.2023.10456393.

[14] D. de-Fitero-Dominguez, E. Garcia-Lopez, A. Garcia-Cabot, and J.-J. Martinez-Herraiz, "Enhanced automated code vulnerability repair using large language models," *Engineering Applications of Artificial Intelligence*, vol. 138, p. 109291, Dec. 2024, doi: https://doi.org/10.1016/j.engappai.2024.109291.

[15] I. Joy, J. Wu, and J. He, "GPT Attack for Adversarial Privacy Policies," pp. 173–180, Aug. 2024, doi: https://doi.org/10.1109/bigcom65357.2024.00032.

[16] H. Rashwan, E. M. Gabidulin, and B. Honary, "Security of the GPT cryptosystem and its applications to cryptography," *Security and Communication Networks*, vol. 4, no. 8, pp. 937–946, Oct. 2010, doi: https://doi.org/10.1002/sec.228