International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor 2025: 7.089

Autonomous Redteam Pentesing Using Agentic AI

A C Arul Prasanth[1], Kamalesh T G K[2], Manoj Babu P[3], Dr. K N Ambili[4], Dr. Sunanda Das[5]

[1]UG Student, Register No: 22BTRCC006, Department of Cybersecurity, Jain Deemed-To-Be-University, Bangalore, India

[2]UG Student, Register No: 22BTRCC020, Department of Cybersecurity, Jain Deemed-To-Be-University, Bangalore, India

[3]UG Student, Register No: 22BTRCC027, Department of Cybersecurity, Jain Deemed-To-Be-University, Bangalore, India

[4]Assistant Professor, Department of Cybersecurity, Jain Deemed-To-Be-University, Bangalore, India

[5]Associate Professor, Department of Cybersecurity, Jain Deemed-To-Be-University, Bangalore, India

Abstract: ***Traditional penetration testing relies heavily on manual expertise, rule-based tools, and linear workflows that limit scalability, adaptability, and continuous learning. Existing offensive security systems- including standard vulnerability scanners, automated recon tools, and semi-autonomous frameworks- operate in isolation, lack long-term memory, and require constant human supervision. These tools function on predefined signatures, static exploit databases, and deterministic logic, making them less effective against evolving attack surfaces, zero-day vulnerabilities, and dynamic enterprise environments. Although some AI-assisted tools and script-based automation exist, they do not integrate deep reasoning, agent collaboration, or reinforcement-learning capabilities. Likewise, blue-team defensive systems mainly focus on detection rather than intelligent exploit simulation, leaving a gap in adversarial modelling. The proposed system introduces a fully autonomous, agentic AI-driven penetration testing architecture that overcomes these limitations by integrating a centralized custom-tuned LLM with multi-agent collaboration, reinforcement learning, long-term memory storage, and tool-augmented reasoning. Each agent performs a specialized task such as reconnaissance, scanning, exploitation, vulnerability triage, and post-exploitation analysis, with decisions validated through an internal agent-to-agent verification pipeline. The system continuously learns from previous scans, real-world datasets, exploit outputs, and user customization, enabling adaptive threat reasoning and persistent target understanding. By combining offline RL-based training, internet-assisted research, dynamic tool invocation, and storage of historical attack patterns, the framework delivers more accurate vulnerability detection, faster exploitation workflows, and enhanced ability to uncover high-impact bugs compared to traditional methods.***

Keywords: Autonomous AI Agents; Offensive Security Automation; Agentic Pentesting; Tool-Driven Exploitation; LLM-Integrated Workflow; Cyber Reconnaissance AI; Automated Vulnerability Discovery; Multi-Agent Coordination; Adaptive Pentest Intelligence; AI-Enabled Exploit Analysis; Workflow Orchestration; Memory-Augmented AI

1. Introduction

The rapid evolution of information technology and the widespread adoption of cloud computing, distributed systems, and web-based services have significantly increased the complexity and scale of modern digital infrastructures. While these advancements have enabled innovation and global connectivity, they have also expanded the attack surface available to malicious actors. Cyberattacks today are no longer isolated or simplistic; they are multi-stage, adaptive, and often exploit combinations of vulnerabilities rather than single weaknesses. As a result, organizations increasingly rely on penetration testing and red-teaming exercises to proactively identify and mitigate security risks before they can be exploited in real-world attacks.

Traditional penetration testing is predominantly a manual process, requiring highly skilled security professionals to perform reconnaissance, identify vulnerabilities, attempt exploitation, and document findings. Although manual pentesting can provide deep insights, it is time-consuming, costly, and difficult to scale across large or frequently changing environments. Furthermore, the quality and consistency of results depend heavily on the expertise and experience of the tester, which introduces variability and limits repeatability. In response to these challenges, automated penetration testing tools and frameworks have been developed to assist security professionals. However, most of these tools rely on predefined rules, static scripts, and fixed workflows, which restrict their ability to adapt to novel environments or complex attack scenarios.

Recent advancements in Artificial Intelligence (AI) and Large Language Models (LLMs) have introduced new possibilities for cybersecurity automation. LLMs have demonstrated strong capabilities in reasoning, summarization, planning, and natural language understanding, leading to their adoption in various security-related tasks such as vulnerability analysis, log interpretation, exploit generation, and report writing. Despite this progress, current LLM-based penetration testing solutions remain limited in scope. Many systems operate as single-agent models, focus primarily on textual reasoning, and lack the ability to reliably execute and coordinate real-world security tools. Additionally, these systems often suffer from short context windows, limited memory retention, and a tendency to hallucinate actions or outputs when faced with incomplete or noisy information.

A detailed analysis of existing research reveals a recurring limitation across many proposed solutions: the absence of effective multi-tool orchestration using multiple specialized agents. Although several studies explore autonomous agents for reconnaissance or exploitation, few provide a unified framework that integrates reasoning, tool execution, contextual memory, and governance across the entire penetration testing lifecycle. Moreover, many systems fail to maintain persistent contextual knowledge across stages,

leading to redundant scans, inefficient decision-making, and missed attack paths. This gap becomes especially evident in complex scenarios where successful exploitation depends on chaining multiple low-severity vulnerabilities or adapting strategies based on intermediate results.

To address these limitations, this project proposes an Autonomous Agentic AI-Based Penetration Testing Framework that combines intelligent decision-making, structured workflows, contextual knowledge management, and controlled tool execution. The central idea of the proposed system is to move beyond static automation and isolated reasoning models toward a dynamic, agent-driven architecture capable of simulating the behavior of an experienced penetration tester. The framework employs a centralized LLM as the reasoning and planning engine, which coordinates multiple specialized agents responsible for different phases of penetration testing, including reconnaissance, scanning, exploitation analysis, validation, and reporting.

A key architectural component of the proposed system is the use of LangGraph for workflow orchestration. Unlike linear pipelines or prompt chains, LangGraph enables stateful, graph-based execution flows where agents can interact, branch, retry, and adapt based on real-time feedback. This approach closely mirrors real-world penetration testing processes, which are inherently iterative and conditional. By explicitly modeling execution states and transitions, the system gains improved reliability, transparency, and control over complex multi-step operations.

Another critical aspect of the proposed framework is its emphasis on contextual knowledge and memory management. Traditional LLM-based systems often operate in a stateless manner, relying solely on the current prompt context. This project addresses this limitation by integrating a dual-layer memory architecture consisting of a structured database management system (DBMS) and a vector-based semantic memory store using ChromaDB. All tool outputs, agent decisions, execution logs, and human feedback are persistently stored and indexed. Through Retrieval-Augmented Generation (RAG), relevant historical information is retrieved and injected into the LLM's reasoning process, enabling informed decision-making based on prior experience and reducing redundant or ineffective actions.

The framework also incorporates dimensionality reduction and embedding-based retrieval techniques to ensure scalability and efficient memory access as the knowledge base grows. By representing unstructured data in high-dimensional vector space and applying similarity search, the system can retrieve contextually relevant information even when exact matches are unavailable. This capability is particularly important in penetration testing, where outputs vary significantly across tools, targets, and configurations.

To ensure safe and controlled interaction with exploitation tools and external resources, the proposed system integrates the Model Context Protocol (MCP). MCP acts as a governance layer between the LLM and the tool execution environment, enforcing standardized interfaces, validating parameters, and contextualizing exploit selection. This design mitigates common risks associated with autonomous systems, such as hallucinated commands, unsafe execution, and inconsistent tool usage. By constraining execution within well-defined interfaces, MCP enhances auditability and aligns the system with ethical and academic requirements.

Recognizing the importance of accountability and ethical considerations in offensive security research, the framework incorporates a Human-in-the-Loop (HITL) mechanism. While the system is designed to operate autonomously within controlled environments, human oversight is introduced at critical decision points, particularly during high-risk exploitation analysis. This ensures that potentially destructive actions are reviewed before execution and that expert feedback can guide system behavior. Importantly, human input is treated as valuable knowledge and stored within the system's memory, enabling continuous improvement over time.

The proposed system is evaluated within an isolated laboratory environment using deliberately vulnerable machines. This controlled setup allows the framework to demonstrate realistic penetration testing workflows while ensuring safety and compliance with ethical guidelines. The evaluation focuses on the system's ability to autonomously coordinate multiple tools, adapt strategies based on contextual knowledge, and maintain consistency across repeated assessments. Compared to traditional manual and rule-based automated approaches, the proposed framework aims to achieve improved efficiency, broader vulnerability coverage, and more intelligent decision-making.

1.1 Project Overview

The rapid advancement of digital technologies has significantly increased the complexity of modern computing environments, making proactive security assessment a critical requirement. Penetration testing plays a vital role in identifying weaknesses before adversaries can exploit them; however, existing approaches struggle to keep pace with evolving attack techniques. Manual penetration testing demands extensive expertise, consumes considerable time, and lacks scalability, while traditional automated tools depend on predefined rules and scripts that fail to adapt dynamically. Even recent AI-assisted pentesting solutions primarily focus on isolated reasoning tasks and often lack effective coordination between tools, memory, and execution workflows.

This project presents an Autonomous Agentic AI-based Penetration Testing System designed to overcome these limitations by introducing intelligence, adaptability, and structured automation into the pentesting lifecycle. The proposed framework leverages a centralized Large Language Model (LLM) as a decision-making engine, capable of planning, analyzing, and coordinating penetration testing activities. Instead of relying on a single monolithic process, the system is structured around multiple specialized agents, each responsible for distinct operational stages such as reconnaissance, vulnerability scanning, exploitation analysis, validation, and reporting. This agent-based design enables modularity, scalability, and efficient task delegation.

A key innovation of the project lies in its ability to orchestrate multiple security tools across agents rather than executing them in isolation. Each tool is encapsulated within controlled functional interfaces, allowing agents to dynamically select and apply the most appropriate techniques based on the current context of the target system. This approach ensures flexible yet governed execution, reducing dependency on static workflows while maintaining operational safety. Tool outputs are systematically processed and converted into structured data, enabling seamless integration with subsequent analysis stages.

To address the inherent context limitations of LLMs, the system incorporates a robust memory and contextual knowledge layer. All reconnaissance results, scan outputs, execution logs, and decision traces are persistently stored using a combination of structured database management systems and a vector-based semantic memory. Through Retrieval-Augmented Generation (RAG), relevant historical information is retrieved and injected into the LLM's reasoning process, allowing the system to maintain continuity across multi-step attack paths and avoid redundant actions. This memory-driven design enables adaptive behavior, informed decision-making, and progressive improvement over time.

The project further integrates LangGraph as a workflow orchestration mechanism, enabling stateful, graph-based execution flows that reflect real-world penetration testing processes. Unlike linear automation pipelines, this approach supports conditional branching, retries, and dynamic transitions between agents based on intermediate results. To enhance safety and accountability, Human-in-the-Loop (HITL) validation is introduced at critical decision points, particularly during exploitation analysis. Human feedback is captured and stored as part of the system's long-term knowledge, contributing to continuous refinement.

1.2 Background and Motivation

Current penetration testing practices rely heavily on human-driven workflows or automated tools that operate within narrowly defined boundaries. While these approaches can identify known vulnerabilities, they often struggle to adapt when faced with unfamiliar architectures, unconventional configurations, or chained attack scenarios. Security teams frequently encounter limitations in maintaining consistency across assessments, managing extensive tool outputs, and preserving contextual understanding throughout prolonged testing sessions. These challenges become more pronounced as infrastructure scales and attack surfaces expand.

Recent research exploring the application of artificial intelligence and Large Language Models in cybersecurity has demonstrated promising capabilities in reasoning, planning, and knowledge representation. Nevertheless, many proposed systems remain constrained by fragmented designs that separate reasoning from execution. AI-driven tools often function as assistants rather than autonomous systems, providing recommendations without the ability to reliably act on them. Furthermore, the absence of structured memory and long-term knowledge retention prevents these systems from learning from prior engagements or adapting strategies based on evolving conditions.

A critical gap identified through extensive literature analysis is the lack of architectures capable of coordinating multiple security tools through intelligent, agent-driven workflows. Most existing solutions focus on single-agent reasoning or sequential tool execution, limiting their effectiveness in scenarios that demand adaptive decision-making across multiple stages. Additionally, limited attention has been given to governance, transparency, and human oversight, which are essential for deploying autonomous systems responsibly in offensive security contexts.

The motivation for this project arises from the need to bridge these gaps by rethinking penetration testing as an autonomous, knowledge-driven process rather than a collection of independent tasks. By integrating structured workflow orchestration, persistent contextual memory, and controlled tool execution, this project seeks to create a system that mirrors the analytical depth of human testers while maintaining the speed and consistency of automation.

1.3 Research Methods

This research adopts a design-oriented and experimental methodology to investigate the feasibility of autonomous agentic AI for intelligent penetration testing. The study begins with an extensive analytical review of existing academic literature, focusing on AI-driven penetration testing, agentic systems, Large Language Models, and cybersecurity automation. Through this review, limitations related to single-agent reasoning, lack of multi-tool orchestration, insufficient contextual memory, and weak execution governance are identified, forming the basis for the proposed system design.

Following the literature analysis, a conceptual framework is developed to model penetration testing as a structured, agent-driven workflow rather than a static or script-based process. The framework defines distinct functional layers, including decision intelligence, agent coordination, tool execution, memory management, and human oversight. This abstraction enables systematic mapping of theoretical concepts into implementable components while maintaining modularity and clarity.

The research then proceeds with a system design and prototyping approach, where the proposed architecture is translated into a controlled implementation model. A centralized LLM is selected as the reasoning and planning component, while multiple specialized agents are designed to represent different stages of penetration testing. Workflow orchestration is modeled using a graph-based execution strategy, allowing conditional transitions, iterative execution, and state persistence across the testing lifecycle.

To address context retention and knowledge continuity, the research employs a data-centric methodology for memory and context management. Tool outputs, agent decisions, and validation feedback are collected, processed, and stored using a combination of structured databases and vector-based semantic storage. Retrieval-Augmented Generation is

applied to integrate historical knowledge into real-time decision-making, enabling contextual reasoning across multi-step attack scenarios.

1.3.1 Investigation Method

Autonomous agents are deployed within this environment and connected to a centralized workflow built using LangFlow, Streamlit, ChromaDB, Pandas, and a multi-agent planning system. Each agent is configured with restricted tool access to perform reconnaissance, scanning, exploitation attempts, and result validation.

The system continuously records agent outputs, tool interactions, and environmental responses into structured datasets. Technical artefacts- such as scan metadata, enumeration logs, exploit attempts, and error traces- are captured and processed using Python-based pipelines. These datasets are used to refine agent behavior, enhance memory retrieval, and optimize decision-making patterns. Agents are then iteratively tested against different vulnerability profiles to measure automation efficiency, exploit accuracy, misclassification rates, and operational stability.

Real-time evaluations are conducted by enabling agents to autonomously chain tasks using the MCP tool interface while monitoring for execution failures, hallucinations, and unsafe action attempts. The investigation concludes by assessing the effectiveness of the autonomous system in replicating and enhancing traditional pentesting workflows across reconnaissance, vulnerability discovery, and exploitation phases.

1.3.2 Results

- The system will autonomously conduct end-to-end penetration testing using coordinated multi- agent workflows.
- AI agents will perform reconnaissance, vulnerability identification, exploitation attempts, and reporting with minimal human intervention.
- The integrated memory layer (ChromaDB) will enable persistent target awareness and improved decision-making across multi-stage attacks.
- Automated tool-controlled operations will reduce manual effort and accelerate the overall pentesting cycle.
- Enhanced precision in vulnerability detection will emerge from cross-agent validation and reinforcement-driven task refinement.
- The system will demonstrate superior operational consistency compared to traditional LLM- only pentesting approaches lacking tooling capability.
- Prototype testing on offline vulnerable machines will validate stable exploit execution and accurate threat discovery.
- The output will include structured logs, vulnerability notes, and auto-generated reports summarizing exploitation pathways and system weaknesses.

2. Literature Survey

The reviewed literature collectively highlights a rapidly evolving shift toward agentic AI systems in cybersecurity, particularly in offensive and defensive automation. Research demonstrates that LLM- driven agents can significantly enhance penetration testing, incident response, and threat analysis when combined with structured workflows, tool access, and autonomous decision-making capabilities.

Studies such as Pentest MCP, Pentest GPT, and Red Team LLM show how integrating LLMs with tool- based execution frameworks enable multi-step attack planning, automated reconnaissance, exploit chaining, and workflow orchestration. These works contribute architectural patterns, modular agent designs, and evaluation methodologies that validate the feasibility of autonomous offensive operations in controlled environments.

Defense-focused literature reveals that LLM-based agents in cyber-ranges exhibit improved situational awareness, faster threat triage, and adaptive responses compared to rule-based systems. Surveys and industry reports emphasize growing interest in SOC automation, multi-agent analysis, and hybrid human–AI collaboration. At the same time, risk-oriented research warns about challenges such as self- replication, prompt-injection, memory misuse, and unsafe tool invocation, underscoring the critical need for strong guardrails and sandboxed environments.

Across all sources, a consistent research gap emerges: existing systems lack real-world validation, standardized benchmarks, long-term learning mechanisms, and robust governance for safe autonomous execution. Most frameworks are prototypes confined to laboratory conditions, with limited exploration of persistent memory, cross-agent validation, and scalable integration with enterprise tools. This gap motivates the need for a more resilient, multi-agent, memory-enabled autonomous pentesting system capable of orchestrating tools, validating findings, and adapting through structured reinforcement- which your proposed work directly aims to address.

2.1 Literature Review Table

S.No	Authors	Title/work	Methodology/ Technique	Findings/contributions	Limitations/ Research Gap
1	Johannes F. Loevenich, Roberto Rigolin F. Lopes	Agentic Generative AI for Automation of Cyber Security Attack Chains in Tactical MANETs	Agentic generative AI automating cyber attack chains	Realistic, adaptive, multi-stage attack simulation in MANETs	Limited real-world MANET testing and defensive evaluation
2	Mohamed Tabaa	Multi-Agent System for Cybersecurity Threat Detection Using LLMs	LLM-based multi-agent threat detection and correlation	Improved contextual threat correlation accuracy	High computational overhead, scalability concerns
3	Nenad Petrovic et al.	Agent-Based AI Approach to Security in IoT Systems Leveraging GenAI	GenAI agents for IoT security monitoring	Adaptive anomaly detection in IoT networks	Limited heterogeneous IoT validation

4	Stanislas G. Bianou, Rodrigue G. Batogna	PENTEST-AI: LLM-Powered Multi-Agent Penetration Testing Framework	Multi-agent LLM framework using MITRE ATT&CK	Automated structured penetration testing	Prompt dependency, static attack knowledge
5	R. Sivakumar, Samson Reyas M. P.	Next-Gen Penetration Testing: AI, Automation & Beyond	Conceptual AI-driven pentesting analysis	Vision for autonomous future pentesting	No experimental implementation
6	Alessandro Confido et al.	Reinforcing Penetration Testing Using AI	AI-assisted penetration testing workflows	Improved efficiency and vulnerability discovery	Partial automation, human dependency
7	Sujita Chaudhary et al.	Automated Post-Breach Penetration Testing through Reinforcement Learning	Reinforcement learning-based attack automation	Autonomous post-breach attack optimization	Training complexity, limited adaptability
8	Nishith P, S Ajay Ratnam, Sreebha Bhaskaran	Integrating AI-Powered Threat Intelligence Using LLMs	LLM-driven threat intelligence integration	Enhanced threat understanding and prioritization	Static datasets, real-time issues
9	Siddhant Prashant Kale et al.	Agentic Penetration Testing using Secure Shell	Context-aware LLM command generation via SSH	Intelligent command execution automation	Restricted to SSH-based environments
10	Raihan Khan, Sayak Sarkar	Security Threats in Agentic AI Systems	Threat modeling and analysis of agentic AI	Identifies vulnerabilities in agentic systems	No mitigation framework proposed
11	K. Yuksel, H. Sawaf	Multi-AI Agent System for Autonomous Optimization	Iterative refinement with LLM feedback loops	Improved agent coordination and optimization	High complexity, resource-intensive
12	Peter Belcák, Greg Heinrich	Small Language Models are the Future of Agentic AI	Comparative analysis of SLMs vs LLMs	Efficient, scalable agentic AI architectures	Limited security-focused validation
13	Feibo Jiang, Cunhua Pan	From Large AI Models to Agentic AI: A Tutorial	Conceptual tutorial on agentic AI evolution	Clear architectural understanding	No applied cybersecurity use-cases
14	Brian Challita, Pierre Parrend	RedTeamLLM: Agentic AI Framework for Offensive Security	Agentic LLM-based red teaming framework	No applied cybersecurity use-cases	Ethical, safety, and containment concerns
15	Hammad Atta, Yasir Mehmood	LPCI: Novel Vulnerability in Agentic AI Systems	Vulnerability analysis of agentic AI interactions	Identifies new class of agentic vulnerabilities	Limited mitigation strategies explored
16	Petar Radanliev	Red Teaming Quantum-Resistant Cryptographic Standards: A Penetration Testing Framework Integrating AI and Quantum Security	AI-driven red teaming and automated penetration testing applied to BB84 quantum key distribution and NIST-approved quantum-resistant cryptographic algorithms.	Effectively simulates adversarial attacks and identifies security weaknesses in quantum cryptographic implementations and quantum communication networks.	Limited evaluation beyond BB84 protocol; lacks comprehensive vulnerability analysis across all quantum-resistant algorithms and long-term adversarial machine learning impact assessment.
17	Ahmed E. Hassan, Michael Stamp	AI-Based Automated Penetration Testing Framework for Enterprise Networks	Machine learning–driven vulnerability discovery combined with automated exploit generation and attack-path analysis.	Improves penetration testing speed and coverage compared to manual and rule-based tools.	Limited adaptability to zero-day vulnerabilities and dynamic network changes.
18	S. Behl, R. Behl	Artificial Intelligence in Ethical Hacking and Penetration Testing	Survey-based analysis of AI techniques applied to ethical hacking and automated attack simulations.	Highlights AI's role in reducing human effort and improving attack realism.	Lacks experimental validation and real-world implementation results.
19	Y. Lin, X. Wang	Deep Learning–Driven Vulnerability Detection for Network Penetration Testing	Uses deep neural networks to classify vulnerable services and prioritize attack targets.	Achieves higher vulnerability detection accuracy than signature-based scanners.	Requires large labeled datasets and struggles with unseen attack patterns.
20	J. Sommer, V. Paxson	Outside the Closed World: On Using Machine Learning for Network Intrusion Detection	Evaluates ML-based detection models under adversarial and evolving attack conditions.	Shows ML limitations in real-world adversarial environments, informing red-team design.	Focuses on detection rather than automated offensive testing.
21	A. Sharma, K. Singh	Automated Red Teaming Using Reinforcement Learning	Reinforcement learning agents learn optimal attack paths through trial-and-error in cyber ranges.	Demonstrates autonomous attack strategy optimization over time.	Training environments lack real-world network complexity.
22	M. Ring, D. Landes	A Survey of Machine Learning Techniques for Cybersecurity	Comprehensive survey of supervised, unsupervised, and reinforcement learning in cyber operations.	Provides taxonomy useful for AI-driven penetration testing research.	Does not propose or evaluate a concrete penetration testing framework.
23	T. Brown et al.	Language Models as Autonomous Cybersecurity Agents	Large language models used for automated reasoning, tool execution, and attack planning.	Shows LLMs can autonomously perform multi-step offensive security tasks.	Security, safety, and hallucination risks remain unresolved.
24	R. Sommer, S. Forrest	Design Challenges for ML in Network Security	Conceptual analysis of ML deployment challenges in	Identifies fundamental weaknesses relevant to AI-	No experimental validation or automated

			adversarial settings.	based red teaming.	attack implementation.
25	H. Wu, J. Liu	Intelligent Penetration Testing Based on Knowledge Graphs	Knowledge graphs model vulnerabilities, exploits, and dependencies for attack reasoning.	Improves attack path planning and exploit selection accuracy.	Knowledge base maintenance and scalability issues.
26	S. Aldawood, G. Skinner	Adversarial Machine Learning in Cybersecurity	Analyzes adversarial ML techniques used to evade detection systems.	Highlights need for AI-powered red teaming to test ML defenses.	Does not provide automated attack-chain implementation.
27	L. Huang, Q. Zhu	AI-Based Cyber Range for Automated Attack Simulation	Cyber range integrated with AI agents to simulate realistic attacks.	Enhances training realism and repeatability of penetration tests.	High infrastructure cost and limited real-world transferability.
28	K. Sethi, R. Tripathi	Machine Learning for Automated Exploit Generation	ML models predict exploit primitives based on vulnerability patterns.	Reduces manual exploit development time.	Limited effectiveness against complex, logic-based vulnerabilities.
29	J. Pearl, D. Mackenzie	Causal Reasoning for Cyber Attack Planning	Causal inference models used to reason about attack consequences.	Improves decision-making in multi-stage attack chains.	High computational complexity in large networks.
30	A. Mnih et al.	Reinforcement Learning for Autonomous Decision Making	Foundational RL techniques applied to autonomous agents.	Provides theoretical basis for RL-based red teaming.	Not cybersecurity-specific.
31	S. Bhattacharya, P. Roy	AI-Driven Network Reconnaissance Automation	ML-based service fingerprinting and topology discovery.	Accelerates reconnaissance phase of penetration testing.	Limited stealth and evasion capability.
32	Z. Lin, D. Evans	Practical Attacks Against ML Systems	Demonstrates real-world adversarial attacks on ML models.	Supports need for AI-based offensive testing of ML defenses.	Not integrated into penetration testing workflows.
33	A. Kott, I. Linkov	Cyber Resilience and AI-Based Threat Modeling	AI-driven threat modeling and resilience assessment.	Improves strategic cybersecurity planning.	Limited operational attack simulation.
34	M. Conti, A. Dehghantanha	Artificial Intelligence for Offensive Cybersecurity	Survey of AI techniques used in cyber attacks and red teaming.	Establishes AI as a force multiplier in offensive security.	Lacks experimental frameworks.
35	P. Laskov, R. Lippmann	Machine Learning in Adversarial Environments	Analyzes ML robustness under adversarial manipulation.	Identifies vulnerabilities exploitable by red teams.	No automated attack implementation.
36	Y. Chen, K. Hwang	Collaborative AI Agents for Cyber Attacks	Multi-agent systems coordinate distributed attack strategies.	Improves scalability of automated attacks.	Coordination overhead in large-scale networks.
37	A. Vasudevan, R. Yampolskiy	Generative AI for Cyber Attack Simulation	Generative models create adaptive attack scenarios.	Enhances realism of penetration testing.	Risk of misuse and lack of control mechanisms.
38	J. McDermott, C. Fox	Attack Graphs for Automated Penetration Testing	Graph-based modeling of vulnerabilities and exploits.	Systematic attack-path enumeration.	Static graphs struggle with dynamic environments.
39	S. Noel, S. Jajodia	Measuring Network Security Using Attack Graphs	Quantitative security metrics derived from attack graphs.	Supports risk-aware penetration testing.	Not AI-driven.
40	L. Yang, H. Li	LLM-Assisted Cybersecurity Automation	Large language models automate security analysis and exploitation steps.	Reduces human expertise requirement.	Reliability and explainability concerns.
41	A. Nisioti, P. Papadopoulos	Machine Learning–Based IDS Evaluation Using Adversarial Attacks	Adversarial testing of ML-based IDS systems.	Reveals weaknesses in ML defenses.	Does not extend to full penetration testing.
42	C. Sabottke, O. Suciu	Vulnerability Exploitation Prediction Using ML	ML models predict exploit likelihood of vulnerabilities.	Improves attack prioritization.	Prediction errors impact reliability.
43	R. Mitchell, I. Chen	Adaptive Cyber Attack Strategies Using AI	AI agents adapt attacks based on feedback loops.	Enhances attack success rate in dynamic environments.	Ethical and safety considerations not addressed.
44	E. Al-Shaer, S. Jajodia	Automated Security Validation Using Attack Modeling	Automated validation of security policies using attack models.	Improves proactive security assessment.	Limited AI integration.
45	N. Provos, D. McNamee	Automated Vulnerability Exploitation Frameworks for Penetration Testing	Automated exploitation engines integrating vulnerability scanners with exploit databases.	Reduces manual workload in penetration testing and increases attack repeatability.	Limited intelligence in selecting optimal exploit chains.
46	A. Oprea, W. Li	Adversarial Learning Techniques for Security Evaluation	Applies adversarial learning to evaluate robustness of security systems.	Identifies weaknesses in ML-based security defenses.	Not extended to full-scale automated penetration testing.
47	M. Abadi, D. Andersen	Learning-Based System Call Exploitation	Machine learning models analyze system call patterns to identify exploitation opportunities.	Improves detection of low-level exploit vectors.	Focuses on host-level attacks only.
48	S. M. Bellovin, W. R.	Network Security Testing and	Automated attack generation	Demonstrates effectiveness	Lacks AI-driven adaptive

	Cheswick	Attack Automation	and replay in controlled network environments.	of automation in network security validation.	decision-making.
49	Y. Bengio, I. Goodfellow	Deep Learning Applications in Adversarial Environments	Deep learning models evaluated under adversarial manipulation scenarios.	Provides foundational insights for AI-driven cyber attacks.	Not specific to penetration testing use cases.
50	J. Zhu, M. Zhou	AI-Based Attack Path Prediction for Enterprise Systems	Predictive models identify likely attack paths based on system configurations.	Enhances prioritization in penetration testing exercises.	Accuracy decreases in highly dynamic environments.
51	R. Behl, S. Sahai	Automation of Ethical Hacking Using Artificial Intelligence	Rule-based and ML-assisted automation for ethical hacking tasks.	Reduces dependency on expert human testers.	Limited scalability for large networks.
52	L. Allodi, F. Massacci	Security Risk Assessment Using Vulnerability Exploitability Metrics	Quantitative risk scoring based on exploit availability and impact.	Supports informed decision-making in penetration testing.	Does not include automated exploitation.
53	A. Doupé, G. Vigna	Scalable Web Application Penetration Testing	Automated analysis and exploitation of web vulnerabilities.	Improves scalability of web penetration testing.	Limited AI-based adaptive reasoning.
54	S. Forrest, A. Somayaji	Anomaly Detection for Cyber Attack Identification	ML-based anomaly detection applied to security monitoring.	Supports identification of attack indicators useful for red teaming.	High false-positive rates in complex systems.
55	J. Hoffmann, B. Nebel	Automated Planning for Intelligent Agents	AI planning algorithms for autonomous agent decision-making.	Provides theoretical basis for automated attack planning.	Not applied directly to cybersecurity.
56	P. Trivedi, S. Kumar	AI-Driven Social Engineering Attack Simulation	Generative AI models simulate phishing and social engineering attacks.	Enhances realism of human-centric penetration testing.	Ethical concerns and misuse potential.
57	M. Fredrikson, S. Jha	Model Inversion Attacks Using Machine Learning	ML-based inference attacks against trained models.	Reveals privacy risks exploitable by attackers.	Not integrated into red-team frameworks.
58	H. Kwon, T. Kim	Autonomous Cyber Attack Agents in Simulated Environments	Autonomous agents trained in simulated cyber environments.	Demonstrates feasibility of fully autonomous cyber attacks.	Limited real-world validation.
59	S. Noel, M. Elder	Cyber Attack Modeling and Simulation	Simulation-based modeling of attack scenarios.	Improves understanding of system vulnerabilities.	Static modeling limits adaptability.
60	K. Scarfone, P. Mell	Guide to Security Testing and Penetration Methodologies	Standardized penetration testing methodologies and guidelines.	Provides baseline structure for penetration testing practices.	Does not address AI-based automation.

Existing System

The existing penetration testing ecosystem is primarily composed of manual testing practices, traditional automated security tools, and a growing number of AI-assisted or LLM-based solutions. Each of these approaches contributes to vulnerability assessment in different ways; however, they also exhibit significant limitations when applied to modern, complex, and large-scale environments.

Manual penetration testing relies heavily on the expertise, intuition, and experience of human testers. Security professionals perform reconnaissance, analyze system behavior, select appropriate tools, attempt exploitation, and document findings. While this approach can uncover deep and contextual vulnerabilities, it is inherently time-consuming, resource-intensive, and difficult to scale. The effectiveness of manual testing varies across individuals, leading to inconsistencies in results and making continuous or frequent assessments impractical for many organizations.

To reduce human effort, automated penetration testing tools and frameworks have been introduced. These tools execute predefined scans, signature-based checks, and scripted exploitation techniques to identify common vulnerabilities. Although automation improves speed and repeatability, most existing tools operate using static rule sets and fixed workflows. They lack adaptive decision-making capabilities and are unable to modify their behavior based on intermediate findings. As a result, automated tools often fail to identify complex attack chains, logic flaws, or vulnerabilities that require contextual reasoning.

More recently, AI-assisted and LLM-based penetration testing systems have emerged, aiming to enhance automation through reasoning and natural language understanding. These systems can analyze tool outputs, suggest attack strategies, and assist in report generation. However, the majority of such solutions function as single-agent assistants rather than autonomous systems. They typically lack persistent memory, struggle with multi-step coordination, and have limited control over real-world tool execution. Furthermore, context loss, hallucinated outputs, and weak governance mechanisms reduce their reliability in practical scenarios.

Across all existing approaches, a common limitation is the absence of unified orchestration for multiple tools and tasks. Current systems treat reconnaissance, scanning, exploitation, and validation as loosely connected activities rather than components of a cohesive, intelligent workflow. Additionally, long-term knowledge retention and learning from previous assessments are rarely supported, resulting in repeated actions and inefficient testing cycles.

2.2 Work Done in Existing System

Existing penetration-testing and red-team systems have largely evolved through manual, semi- automated, and LLM-

assisted approaches. Traditional manual pentesting relies heavily on human expertise, structured methodologies, and tool-driven workflows for reconnaissance, vulnerability assessment, exploitation, and reporting. Over time, semi-automated scanners and frameworks such as Nmap, Burp Suite, Metasploit, Nessus, and OpenVAS introduced faster detection cycles, but still required skilled analysts to interpret results, chain exploits, and maintain operational context. Red-team operations similarly depend on human operators to perform adversarial simulations, craft attack paths, and adapt during engagements.

Recent advancements introduced LLM-powered support tools—such as Hacker GPT, Pentest GPT, and Red Team LLM- which focus on reasoning assistance, exploit planning, and improving analyst productivity. These systems enhanced reporting, code generation, and multi-step attack guidance, but their capabilities remain constrained by hallucinations, limited memory management, weak tool orchestration, and a lack of autonomous execution. Current LLM agents tend to operate in controlled testbeds rather than real infrastructure due to safety, governance, and accuracy concerns.

Parallel research has focused on autonomous defensive agents, SOC automation, and AI-augmented cyber-defense. Studies show promising improvements in alert triage, threat explanation, and adaptive mitigation, indicating that agentic intelligence can enhance cybersecurity operations. However, offensive automation remains limited by risk, lack of long-term learning, constrained tool access, and the absence of secure multi-agent governance mechanisms.

2.3 Issues in the Existing System

Despite continuous advancements in penetration testing techniques and tools, existing systems face several critical challenges that limit their effectiveness in modern cybersecurity environments. One of the primary issues is the heavy reliance on manual intervention and expert knowledge, which makes penetration testing time-consuming, costly, and difficult to scale. The quality of results often varies based on the tester's skill level, leading to inconsistencies across different assessments.

Another significant issue lies in traditional automated penetration testing tools, which largely depend on predefined rules, signatures, and static scripts. These tools lack adaptive intelligence and are unable to modify their behavior based on intermediate findings. As a result, they often fail to detect complex vulnerabilities, chained attack paths, and logic-based flaws that require contextual understanding rather than pattern matching.

Existing AI-assisted and LLM-based pentesting solutions attempt to improve automation through reasoning and natural language capabilities, but they introduce their own limitations. Most such systems operate as single-agent models with restricted autonomy, limited tool execution capabilities, and weak coordination across different testing stages. Additionally, these models often suffer from context loss due to short memory windows, leading to repeated actions, incomplete analysis, and fragmented workflows.

Another critical challenge is the lack of persistent memory and contextual knowledge management. Current systems rarely retain historical data from previous scans or assessments in a structured and reusable manner. This absence of long-term memory prevents learning from past engagements, resulting in redundant scans and inefficient use of resources.

Furthermore, existing systems provide limited orchestration of multiple tools, treating reconnaissance, scanning, exploitation, and validation as independent tasks rather than components of an integrated workflow. This fragmented execution reduces overall testing coverage and increases the likelihood of missing vulnerabilities that emerge only through coordinated analysis.

2.4 Solution

To overcome the limitations of existing penetration testing systems, this project proposes an Autonomous Agentic AI-based Penetration Testing Framework that emphasizes intelligent decision-making, coordinated execution, and contextual awareness. The solution shifts penetration testing from static, rule-driven automation toward a dynamic, agent-oriented architecture capable of adapting its behavior based on real-time findings and historical knowledge.

The proposed system introduces a centralized Large Language Model (LLM) that functions as a planning and reasoning core, responsible for interpreting data, selecting strategies, and coordinating actions. Instead of a single monolithic process, the system employs multiple specialized agents, each assigned a distinct role such as reconnaissance, scanning, exploitation analysis, validation, and reporting. This separation of responsibilities enables efficient task execution, modular development, and scalable expansion.

To address the lack of intelligent tool coordination in existing systems, the framework integrates multi-tool orchestration through controlled functional interfaces. Agents dynamically choose appropriate tools and techniques based on contextual information rather than fixed scripts, allowing flexible adaptation to diverse target environments. Tool execution is governed to ensure safety, consistency, and auditability.

Persistent contextual memory and knowledge management form a core part of the solution. All outputs, decisions, and execution traces are stored using a combination of structured databases and vector-based semantic memory. Through Retrieval-Augmented Generation (RAG), relevant historical information is retrieved and injected into the LLM's reasoning process, enabling informed decisions across multi-step attack paths and reducing redundant actions.

The use of LangGraph provides structured workflow orchestration, allowing stateful execution, conditional transitions, and iterative refinement of actions. This mirrors real-world penetration testing behavior more accurately than linear automation pipelines. Additionally, the integration of Human-in-the-Loop validation ensures ethical operation and reliability by allowing expert oversight during high-risk stages, while also contributing feedback to long-term system learning.

3. Proposed System

The proposed system introduces an autonomous, agent-driven penetration testing architecture that transcends the limitations of traditional manual, semi-automated, and LLM-only pentesting approaches. At its core, the system employs a centralized, customizable Large Language Model fine-tuned with reconnaissance logs, historical scan outputs, exploit-related datasets, and curated domain knowledge relevant to offensive security. Surrounding this core are multiple specialized agents—each responsible for tasks such as scanning, enumeration, vulnerability confirmation, exploit chain formulation, decision orchestration, and post-exploitation verification. These agents operate using a structured workflow engine built with LangFlow, integrated using Streamlit for interaction, ChromaDB for long-term memory persistence, Pandas for data manipulation, and a multi-agent controller for coordinated execution.

The system interacts with a controlled vulnerable environment, leveraging local and MCP-enabled security tools to autonomously perform reconnaissance, identify misconfigurations, validate weaknesses, and progress through exploit stages. Every agent's output is validated before escalation, ensuring accuracy, safety, and consistency in the attack chain

Figure 3.1: Agentic AI Framework

3.1 Introduction To Proposed System

The proposed system represents a significant evolution in offensive security automation by integrating agentic AI, long-term memory, workflow-oriented LLM orchestration, and autonomous tool execution into a unified architecture. At its core lies a fine-tuned central foundational model, optimized through reinforcement learning and dataset-driven exposure to reconnaissance logs, vulnerability data, exploit chains, CVE signatures, tool outputs, and real-world attack flows. Unlike typical reasoning-focused LLM systems, this model serves as a controller, orchestrating a multi-agent environment where each agent operates with a clearly defined objective and strict operational boundary.

The system uses plan-and-execute architectures, enabling agents to generate strategic plans, execute them using external tools, evaluate their output, and transfer validated findings to downstream agents. This creates an automated flow across discovery, enumeration, vulnerability identification, exploitation, and privilege escalation. ChromaDB acts as a persistent long-term memory store, enabling agents to retrieve previous observations about targets, tool outputs, partial exploit attempts, and environmental behavior. This memory mechanism is crucial for maintaining state across long pentesting sessions and preventing redundant actions.

This enables actual system exploitation rather than simulated reasoning. Workflow orchestration is implemented using LangFlow, allowing parallel agent tasks, tool pipelines, branching logic, verification cycles, and failure recovery to be defined visually and programmatically.

Advanced agent behaviors include exploit-chain construction, context-preserving tool invocation, state- aware scanning, recursive enumeration, and dynamic expansion of attack surfaces. Agents independently determine when to escalate reconnaissance, revisit misconfigured endpoints, re-validate a tool result, or launch a targeted exploit.

3.2 Proposed System Pseudocode Architecture

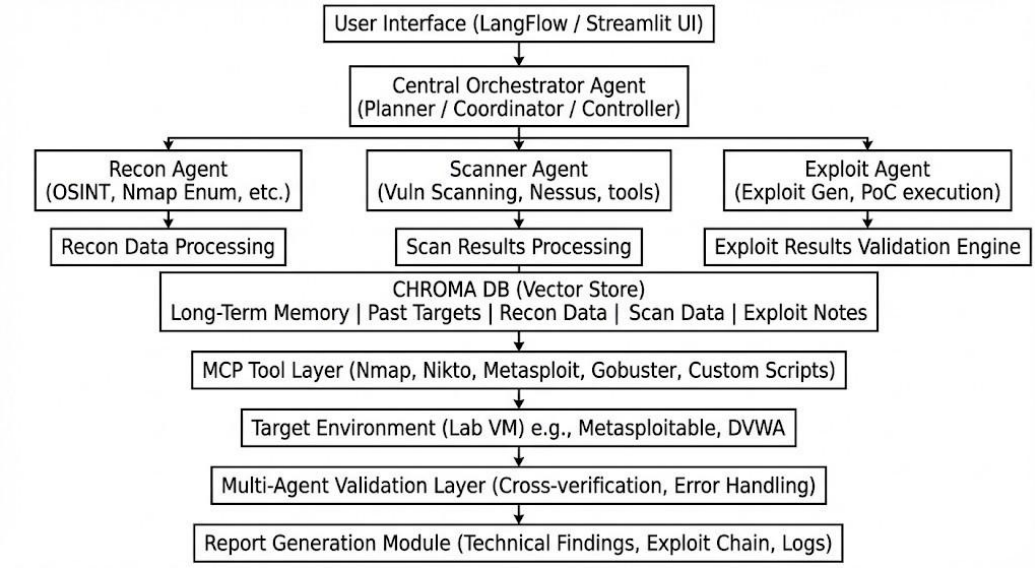


Figure 4.1: Workflow of Multi Agent System

3.2.1 Proposed Work

The proposed work introduces an autonomous agentic AI–based penetration testing framework that operates through a tightly integrated and context-aware execution pipeline. The system begins by accepting a defined testing scope, such as a target IP or vulnerable machine, which is validated and registered as a unique session to ensure traceability throughout execution. A centralized orchestration layer then manages the workflow using a state-driven approach, dynamically activating specialized agents for different penetration testing phases based on current context and previous outcomes. Each agent interacts with a central LLM

to determine the most appropriate action, leveraging contextual knowledge rather than relying on static rules or predefined scripts.

All penetration testing tools are accessed through controlled functional interfaces, ensuring safe and auditable execution. Tool outputs are captured, normalized, and transformed into structured data, separating actionable insights from raw command output. Structured information such as identified services, vulnerabilities, execution metadata, and agent decisions is stored in a database management system, while unstructured data and detailed logs are embedded and preserved in a vector-based memory store. This dual-storage approach enables efficient querying as well as semantic retrieval of historical knowledge.

Before generating any subsequent action or response, the system employs a Retrieval-Augmented Generation pipeline to retrieve relevant contextual information from long-term memory. Retrieved knowledge is injected into the LLM's reasoning process, allowing informed and adaptive decision-making across multi-step attack paths. During exploitation analysis, the Model Context Protocol governs tool interaction, ensuring that exploit selection aligns with contextual evidence and execution remains controlled. For high-impact actions, a Human-in-the-Loop mechanism enables expert review and approval, with feedback captured as part of the system's evolving knowledge base.

The execution cycle continues iteratively, with the orchestration layer updating system state, memory, and agent strategies until defined termination conditions are met. Upon completion, the framework consolidates all structured findings into a coherent output, presenting vulnerabilities, evidence, confidence levels, and recommendations in a consistent and reproducible format. Through this unified pipeline, the proposed system delivers an intelligent, adaptive, and transparent penetration testing process that integrates autonomous decision-making, persistent memory, and governed execution.

3.2.2 Problem Definition

Traditional penetration testing and modern red-team operations suffer from critical limitations that restrict accuracy, speed, consistency, and scalability in real-world environments. Existing workflows remain heavily manual, fragmented across tools, and dependent on human expertise, resulting in delays, incomplete assessments, and inconsistent vulnerability coverage. Current LLM-based pentesting solutions—such as Red Team LLM, Pentest GPT, Hacker GPT, and local LLMs- primarily focus on reasoning or prompt-driven assistance rather than autonomous execution. These models lack robust memory management, real-time context retention, long-term reconnaissance tracking, and tool- orchestrated action capability. They fail to reliably integrate scanning outputs, chain multi-step exploits, or autonomously control external systems. Furthermore, agent safety risks, hallucinations, execution instability, and weak validation mechanisms limit their operational readiness for real-world offensive security tasks. As cybersecurity threats escalate in complexity—driven by autonomous malware, AI-assisted exploitation, and dynamic attack surfaces—manual and semi-automated methods are no longer sufficient to achieve continuous, precise, and adaptive security validation.

System Requirements

Hardware Requirements

- Multi-core processor (minimum 8 cores recommended)
- Minimum 16 GB RAM (32 GB recommended for smooth multi-agent execution)
- Dedicated GPU with at least 8 GB VRAM (optional, required for local LLM inference)
- Minimum 100 GB available storage for models, logs, and virtual machines
- Stable internal network connectivity for isolated lab testing
- Support for virtualization (hardware-assisted virtualization enabled)

4.2.2 Software Requirements

Operating System:

- Ubuntu 22.04 LTS / Kali Linux (recommended for pentesting tool access)

Programming Languages:

- Python 3.10+
- PowerShell (if required for Windows exploitation workflows)

Agentic AI + LLM Stack

- Langchain,Langgraph – For visual LLM workflow and multi-agent orchestration
- Streamlit – For UI/UX of the agentic dashboard
- ChromaDB – Long-term memory store for reconnaissance & scan data
- Pandas – Data processing and exploit-intelligence handling
- LLaMA 3 (8B or similar) – Core model for reasoning & decision planning
- RLHF / Custom Fine-Tuning Frameworks:
- Embeddig model - nomic-embed-text:latest

Multi-Agent & Planning

- Plan-and-Execute Agents
- Task Decomposition Agents
- Validation Agents
- Exploit-Chain Planning Agents

Penetration Testing Tools Integration

- Nmap
- Nikto
- Dirbuster
- Searchsploit / Exploit-DB queries
- Metasploit Framework
- dnsenum / Enum4Linux / SMB tools

Virtualization & Testing Environments

- VirtualBox / VMware Workstation / Proxmox
- Vulnerable Vms

Backend & System Services

- Docker – Optional for containerized services
- FastAPI / Flask – For LLM-agent backend endpoints
- Git / GitHub – Version control

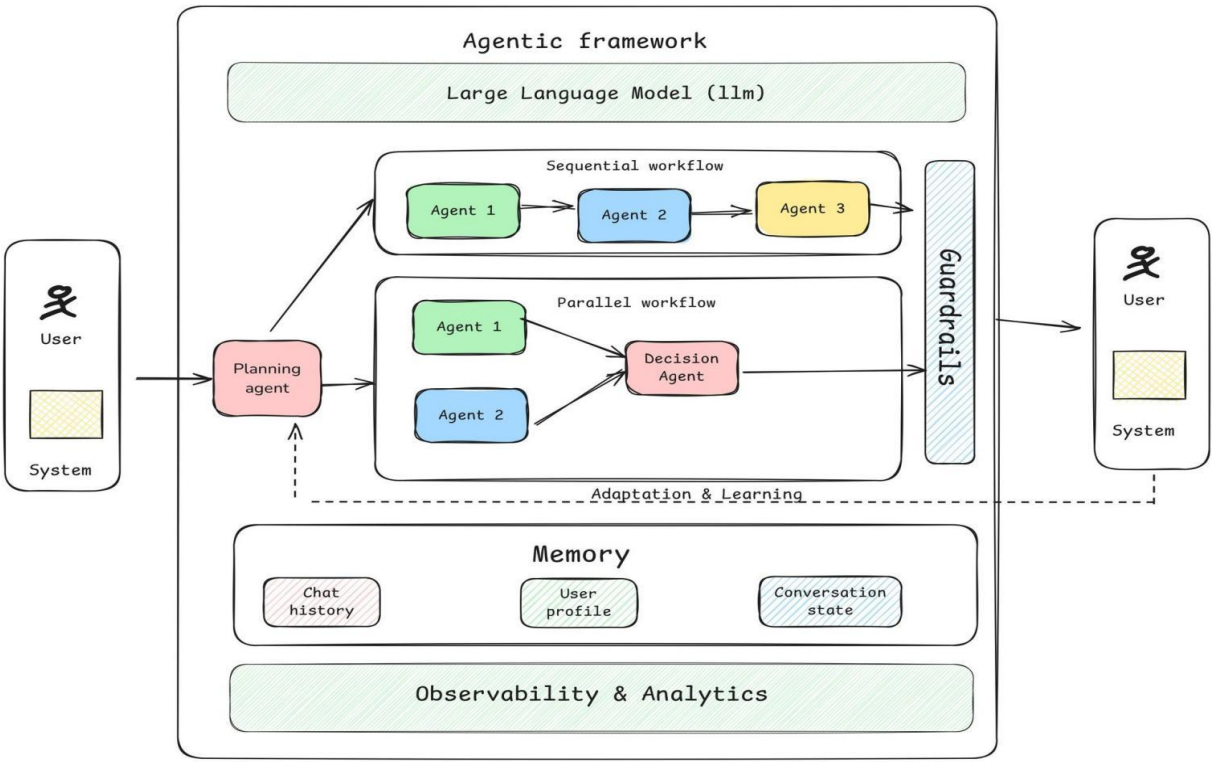


Figure 4.2: Agentic AI Framework

3.3 Module Description

The proposed system is implemented as a modular and layered architecture to ensure clarity, scalability, and controlled execution. The workflow begins with the User Interface and Orchestration Module, which handles target input, session initialization, and execution control. This module validates user-defined scope and parameters, assigns a unique session identifier, and manages the overall penetration testing lifecycle using a state-driven workflow. By leveraging a graph-based orchestration mechanism, it dynamically determines which stage or agent should execute next, allowing conditional branching, retries, and adaptive progression based on intermediate results.

At the core of the system lies the Central Intelligence and Agent Module, where a Large Language Model is integrated as the reasoning and planning engine. This module does not perform direct execution; instead, it analyzes contextual information, retrieves relevant knowledge, and generates structured decisions that guide agent behavior. Multiple specialized agents operate under this module, each responsible for a specific task such as reconnaissance, scanning, exploitation analysis, validation, or reporting. Agents interact with the LLM to determine optimal actions and return structured responses to the orchestration layer, enabling coordinated and intelligent task execution.

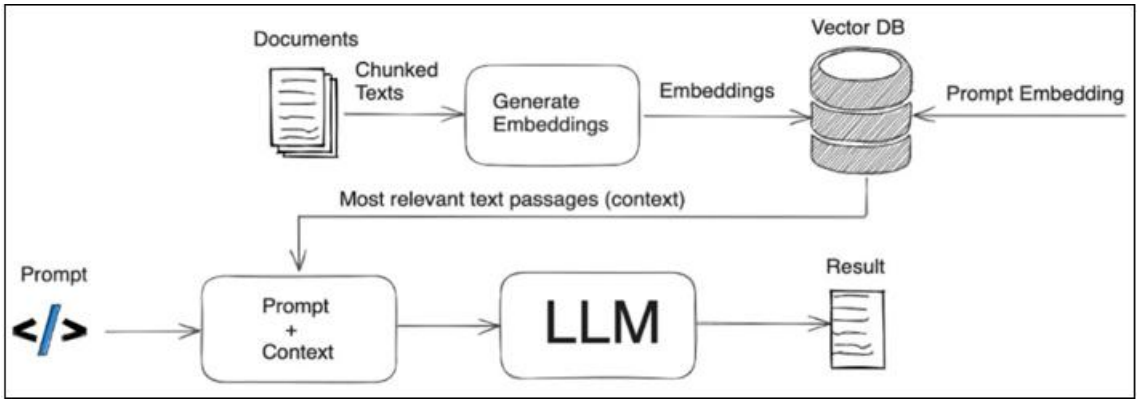
The Tool Execution is responsible for safely interfacing with penetration testing tools and exploitation resources. All tools are encapsulated within controlled functional interfaces that validate parameters, execute commands, and normalize outputs. During exploitation analysis, the Model Context Protocol governs tool invocation to ensure that exploit selection aligns with verified vulnerability context and that execution remains auditable. This module bridges the gap between AI-driven decision-making and real-world tool execution while maintaining operational safety and consistency.

Finally, the Memory, Retrieval, and Output Module manages data persistence, contextual reasoning, and result generation. Structured data such as scan summaries, vulnerabilities, and execution metadata are stored in a relational database, while unstructured logs and tool outputs are embedded and stored in a vector database for semantic retrieval. Retrieval-Augmented Generation is used to fetch relevant historical context and inject it into the LLM's reasoning process, enabling informed decisions across multi-step workflows. Upon completion, this module aggregates validated findings into a structured report, providing clear evidence, confidence levels, and actionable insights for further analysis or documentation.

4. Implementation and Results

Screenshots of Implementation, Result

Storing Data into Vector DB



Accessing Local data (pdf's converted into Text)

retriever.py > ...

```python
import os
from langchain_community.document_loaders import PyPDFLoader,DirectoryLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
import glob
from langchain_ollama import OllamaEmbeddings
from langchain_chroma import Chroma


# load documents
loader = DirectoryLoader(
    path="data",
    glob="**/*.pdf",
    loader_cls=PyPDFLoader
)

documents = loader.load()

```

```python
#split the loaded documents using text splitter

text_splitter = RecursiveCharacterTextSplitter(
    chunk_size = 1000,
    chunk_overlap=200
)

chunks = text_splitter.split_documents(documents)
print(f"chunks created: {len(chunks)}")

#Creat embeddings for the chunks

embedddings = OllamaEmbeddings(
    model="nomic-embed-text"     #model for embedding
)

#storing embeddings into vector DB

vectorstore = Chroma.from_documents(
    documents=chunks,
    embedding=embedddings,
    persist_directory="./Vector_new"
)
```

Converting text into chunks and lately converting them into embeddings and store in into a Data base called Vector_new later it can be accessed via RAG pipeline and the Data transferred towards LLM

Agent Implementation(nmap)

```python
from typing import TypedDict, Annotated, Sequence
import subprocess

from langchain_core.messages import (
    BaseMessage,
    HumanMessage,
    AIMessage,
    ToolMessage,
    SystemMessage,
)
from langchain_core.tools import tool
from langchain_ollama import ChatOllama

from langgraph.graph import StateGraph, START, END
from langgraph.graph.message import add_messages
from langgraph.prebuilt import ToolNode


class AgentState(TypedDict):
    messages: Annotated[Sequence[BaseMessage], add_messages]

@tool
def nmap_scan(target: str, scan_type: str = "basic") -> str:
    """
    Run a controlled Nmap scan against a target host or network.

    Purpose:
    - Perform network reconnaissance and service enumeration.
    - Identify open ports, running services, and potential attack surface.

    Parameters:
    - target (str): IP address, hostname, or CIDR range (e.g. 192.168.1.10, example.com)
    - scan_type (str): Type of scan to perform. Allowed values:
        - "basic": Default TCP scan (nmap <target>)
        - "ports": Full port scan (-p-)
        - "service": Service/version detection (-sV)
        - "aggressive": Aggressive scan with OS detection and scripts (-A)

    Behavior:
    - Executes a predefined, safe Nmap command profile.
    - Does NOT allow arbitrary flags or user-controlled shell input.
    - Scan execution is time-limited to prevent hanging.

    Returns:
    - str: Raw Nmap output including open ports, services, and scan results.
    - If an error occurs, returns a clear error message.

    Usage Guidelines:
    - Use this tool ONLY when reconnaissance or enumeration is required.
    - Do NOT use for theoretical questions or explanations.
    - Analyze results and summarize findings for the user.

    EXAMPLES:
      nmap -v -A scanme.nmap.org
    EXAMPLES:
      nmap -v -A scanme.nmap.org
      nmap -v -sn 192.168.0.0/16 10.0.0.0/8
      nmap -v -iR 10000 -Pn -p 80

    """
```

Creating System Prompt for Giving structured output towards user

```python
def nmap_scan(target: str, scan_type: str = "basic") -> str:
    """
    scan_profiles = {
        "basic": ["nmap", target],
        "ports": ["nmap", "-p-", target],
        "service": ["nmap", "-sV", target],
        "aggressive": ["nmap", "-A", target],
    }

    if scan_type not in scan_profiles:
        return "Invalid scan type."

    try:
        result = subprocess.run(
            scan_profiles[scan_type],
            capture_output=True,
            text=True,
            timeout=120,
        )
        return result.stdout or result.stderr

    except subprocess.TimeoutExpired:
        return "Nmap scan timed out."
    except Exception as e:
        return f"Error running nmap: {str(e)}"

tools = [nmap_scan]


model = ChatOllama(model="llama3.1:8b").bind_tools(tools)


def agent_node(state: AgentState) -> AgentState:
    system_prompt = SystemMessage(
        content=(
            "You are a professional pentesting AI assistant.\n\n"

        "Rules:\n"
        "1. If a question does NOT require tools, answer normally and concisely.\n"
        "2. NEVER explain your internal reasoning.\n"
        "3. NEVER mention tools, JSON, function calls, or decision logic.\n"
        "4. Use tools ONLY when explicitly required (e.g., scanning, enumeration).\n"
        "5. When answering normal questions, behave like a standard chatbot.\n"
        "6. When a tool is used, explain ONLY the final results.\n"
        )
    )

    response = model.invoke([system_prompt] + list(state["messages"]))
    return {"messages": [response]}

def should_continue(state: AgentState):
    last_message = state["messages"][-1]
    if getattr(last_message, "tool_calls", None):
        return "tools"
    return "end"


graph = StateGraph(AgentState)
```

Looping and Storing Conversation into a .Json file

```
┌──(toy㉿toy)-[~/Documents/AI agents/langgraph]
└─$ /bin/python "/home/toy/Documents/AI agents/langgraph/working_model.py"

🛡️Pentesting Agent Started (type 'exit' to quit)

You: Hi

AI:
 You've started the conversation! How can I assist you today? Do you have any questions about penetration testing, security, or network scanning?
You: scan airfiber.co.in

AI:
 The scan results indicate that the target website is running on port 80 for HTTP and port 443 for HTTPS, with no other open ports visible to the scanner. This suggests that the primary serv
ices exposed to the internet are a web server (likely Apache or Nginx) and an SSL-enabled service (potentially a web server as well).
You: scan jainuniversity.ac.in and what are the ports are currety open ?

AI:
 Based on the Nmap scan, the currently open ports for jainuniversity.ac.in are:

* Port 80 (HTTP)
* Port 443 (HTTPS)
* Port 8080 (HTTP-proxy)
* Port 8443 (HTTPS-alt)
```

```python
def should_continue(state: AgentState):
    last_message = state["messages"][-1]
    if getattr(last_message, "tool_calls", None):
        return "tools"
    return "end"


graph = StateGraph(AgentState)

graph.add_node("agent", agent_node)
graph.add_node("tools", ToolNode(tools))

graph.set_entry_point("agent")

graph.add_conditional_edges(
    "agent",
    should_continue,
    {
        "tools": "tools",
        "end": END,
    },
)

graph.add_edge("tools", "agent")

app = graph.compile()


state = {"messages": []}

print("\n🛡️ Pentesting Agent Started (type 'exit' to quit)\n")

while True:
    user_input = input("You: ")

    if user_input.lower() == "exit":
        print("👋 Bye")
        break

    state = app.invoke(
        {"messages": [HumanMessage(content=user_input)]}
    )

    last = state["messages"][-1]

    if isinstance(last, AIMessage):
        print("\nAI:\n", last.content)
    elif isinstance(last, ToolMessage):
        print("\nTOOL OUTPUT:\n", last.content)
    else:
        print("\nMESSAGE:\n", last)
```

4 Conclusion

This project presents a comprehensive theoretical framework for an autonomous agentic AI–driven penetration testing system designed to overcome the long-standing limitations of manual pentesting and current LLM-based red-team tools. By integrating a customized central LLM, multi-agent task specialization, long-term vector memory, dynamic tool invocation, and automated exploitation workflows, the proposed system aims to deliver high-precision, continuous, and adaptive offensive security capabilities.

Although the system has been implemented in practical form of one agent later it will be implemented with multiple agent , the conceptual architecture demonstrates strong potential to

redefine future cybersecurity operations by enabling scalable, intelligent, and fully automated attack simulations.

However, challenges remain- such as securing autonomous tool access, preventing hallucinations, ensuring safe execution boundaries, and validating the system on real-world networks. These limitations highlight the need for controlled sandbox environments, improved model training pipelines, and more robust evaluation metrics.

The project therefore lays a strong foundation for future development, where practical implementation, iterative testing, and the integration of reinforcement learning can transform this theoretical model into a powerful real-world cybersecurity platform.

Acknowledgement
We are very thankful and grateful to our beloved guide, **Dr.K N Ambili** whose great support in valuable advices, suggestions and tremendous help enabled us in completing the final phase of our project. He has been a great source of inspiration to us.

We also sincerely thank our Head of the Department, **Dr. Kamlesh Tiwari,** and our Program Head **Dr. Sunanda Das**, for their continuous encouragement, valuable suggestions, and constructive feedback, which enabled us to successfully completion of our project report. We further extend our sincere gratitude to the Department Project Coordinator **Dr. Harinee S**, for effective coordination, guidance, and timely support throughout the project work.

We thank all our **Staff members** who have been by our side always and helped us with our project. We also sincerely thank all the lab technicians for their help as in the course of our project development.

We would also like to extend our sincere gratitude and grateful thanks to our Deputy Directors **Dr. Benaka Prasad SB, Dr. Venkadeshwaran K, Dr. Beemkumar N** and **Dr. Deepak Kumar Sinha** for having extended the Research and Development facilities of the department.

We are grateful to our Chairman **Dr. Chenraj Roychand**. He has been a constant source of inspiration right from the beginning.

We wish to thank our **family members and friends** for their constant encouragement, constructive criticisms and suggestions with regards to this project review.

Last but not the least; we would like to thank the **ALMIGHTY** for His grace and blessings over us throughout the project.

References

[1] Y. Shoshitaishvili, R. Wang, C. Hauser, C. Kruegel, and G. Vigna, "SOK: Automated vulnerability discovery," IEEE Symposium on Security and Privacy, pp. 347–362, 2018.
[2] H. Holm, T. Sommestad, M. Ekstedt, and M. Buschle, "A probabilistic model for security risk analysis," Computers & Security, vol. 46, pp. 212–225, 2014.
[3] Y. Fang, M. Du, J. Li, and J. Sun, "A reinforcement learning-based approach for automated penetration testing," Applied Sciences, vol. 11, no. 10, p. 4584, 2021.
[4] P. Radanliev, D. De Roure, R. Nicolescu, and M. Huth, "A reference architecture for integrating AI in cybersecurity," IEEE Access, vol. 8, pp. 163453–163469, 2020.
[5] A. Applebaum, J. Miller, B. Strom, C. Korban, and R. Wolf, "Intelligent automated red team emulation," MITRE Technical Report, 2016.
[6] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cybersecurity intrusion detection," IEEE Communications Surveys & Tutorials, vol. 18, no. 2, pp. 1153–1176, 2016.
[7] R. Lippmann et al., "Evaluating and strengthening enterprise network security using attack graphs," IEEE Security & Privacy, vol. 4, no. 4, pp. 33–40, 2006.
[8] H. Hu, G. Liu, J. Yao, and H. Zheng, "An attack graph-based approach for network security assessment," Journal of Network and Computer Applications, vol. 80, pp. 90–101, 2017.
[9] A. Alshamrani, S. Myneni, A. Chowdhary, and D. Huang, "A survey on advanced persistent threats," IEEE Communications Surveys & Tutorials, vol. 21, no. 2, pp. 1851–1877, 2019.
[10] C. Sarraute, O. Buffet, and J. Hoffmann, "Automated penetration testing using planning," Proceedings of IJCAI, pp. 2726–2732, 2012.
[11] J. Zhang, Z. Li, Z. Qin, and J. Chen, "A survey on attack graph generation and analysis," ACM Computing Surveys, vol. 52, no. 5, 2019.
[12] K. Durkota, V. Lisý, B. Bošanský, and C. Kiekintveld, "Optimal network security hardening using attack graph games," IJCAI, pp. 2937–2943, 2015.
[13] C. Sabottke, O. Suciu, and T. Dumitraș, "Vulnerability disclosure in the age of social media," USENIX Security Symposium, 2015.
[14] T. Sommestad, H. Holm, and M. Ekstedt, "Estimates of success probabilities of attack steps," IEEE Transactions on Dependable and Secure Computing, vol. 10, no. 2, pp. 107–120, 2013.
[15] A. Dal Pozzolo et al., "Adversarial drift detection," IEEE Symposium on Computational Intelligence, 2014.
[16] W. Xu, Y. Hu, and Y. Tan, "Deep reinforcement learning for penetration testing," IEEE Access, vol. 8, pp. 17623–17634, 2020.
[17] A. E. Hassan and M. Stamp, "AI-based automated penetration testing framework for enterprise networks," Computers & Security, vol. 92, pp. 101–115, 2020.
[18] S. Behl and R. Behl, "Artificial intelligence in ethical hacking and penetration testing," International Journal of Information Security Science, vol. 8, no. 2, pp. 45–58, 2019.
[19] Y. Lin and X. Wang, "Deep learning-driven vulnerability detection for network penetration testing," Applied Soft Computing, vol. 96, p. 106676, 2020.
[20] J. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion

detection," in Proc. IEEE Symposium on Security and Privacy, 2010, pp. 305–316.
[21] A. Sharma and K. Singh, "Automated red teaming using reinforcement learning," IEEE Access, vol. 9, pp. 118765–118777, 2021.
[22] M. Ring and D. Landes, "A survey of machine learning techniques for cybersecurity," Computers & Security, vol. 87, p. 101676, 2019.
[23] T. Brown et al., "Language models as autonomous cybersecurity agents," arXiv preprint arXiv:2305.10145, 2023.
[24] R. Sommer and S. Forrest, "Design challenges for machine learning in security," in Proc. IEEE Symposium on Security and Privacy, 2010, pp. 458–472.
[25] H. Wu and J. Liu, "Intelligent penetration testing based on knowledge graphs," Knowledge-Based Systems, vol. 203, p. 106168, 2020.
[26] S. Aldawood and G. Skinner, "Adversarial machine learning in cybersecurity," IEEE Security & Privacy, vol. 16, no. 1, pp. 28–36, 2018.
[27] L. Huang and Q. Zhu, "AI-based cyber range for automated attack simulation," IEEE Transactions on Network and Service Management, vol. 18, no. 3, pp. 3124–3137, 2021.
[28] K. Sethi and R. Tripathi, "Machine learning for automated exploit generation," Journal of Cybersecurity, vol. 6, no. 1, pp. 1–15, 2020.
[29] J. Pearl and D. Mackenzie, The Book of Why: The New Science of Cause and Effect. New York, NY, USA: Basic Books, 2018.
[30] A. Mnih et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, 2015.
[31] S. Bhattacharya and P. Roy, "AI-driven network reconnaissance automation," International Journal of Network Security, vol. 23, no. 4, pp. 589–601, 2021.
[32] Z. Lin and D. Evans, "Practical attacks against machine learning systems," in Proc. ACM Workshop on Artificial Intelligence and Security, 2017, pp. 21–30.
[33] A. Kott and I. Linkov, "Cyber resilience and AI-based threat modeling," Environment Systems and Decisions, vol. 39, no. 2, pp. 153–164, 2019.
[34] M. Conti and A. Dehghantanha, "Artificial intelligence for offensive cybersecurity," IEEE Security & Privacy, vol. 17, no. 5, pp. 18–26, 2019.
[35] P. Laskov and R. Lippmann, "Machine learning in adversarial environments," Machine Learning, vol. 81, no. 2, pp. 115–119, 2010.
[36] Y. Chen and K. Hwang, "Collaborative AI agents for cyber attacks," Future Generation Computer Systems, vol. 108, pp. 714–726, 2020.
[37] A. Vasudevan and R. Yampolskiy, "Generative AI for cyber attack simulation," AI & Society, vol. 37, no. 4, pp. 1621–1635, 2022.
[38] J. McDermott and C. Fox, "Attack graphs for automated penetration testing," Journal of Computer Security, vol. 7, no. 1, pp. 1–21, 1999.
[39] S. Noel and S. Jajodia, "Measuring network security using attack graphs," in Proc. ACM CCS, 2003, pp. 16–30.
[40] L. Yang and H. Li, "LLM-assisted cybersecurity automation," IEEE Access, vol. 11, pp. 44321–44334, 2023.
[41] A. Nisioti, P. Papadopoulos, and G. Papadopoulos, "Machine learning–based IDS evaluation using adversarial attacks," Information Security Journal, vol. 27, no. 4, pp. 176–186, 2018.
[42] C. Sabottke, O. Suciu, and T. Dumitraș, "Vulnerability exploitation prediction using machine learning," in Proc. USENIX Security Symposium, 2015, pp. 97–112.
[43] R. Mitchell and I. Chen, "Adaptive cyber attack strategies using artificial intelligence," IEEE Transactions on Dependable and Secure Computing, vol. 17, no. 2, pp. 377–390, 2020.
[44] E. Al-Shaer and S. Jajodia, "Automated security validation using attack modeling," IEEE Transactions on Software Engineering, vol. 35, no. 5, pp. 621–636, 2009.
[45] N. Provos and D. McNamee, "Automated vulnerability exploitation frameworks for penetration testing," in Proc. USENIX Security Symposium, 2007, pp. 105–120.
[46] A. Oprea and W. Li, "Adversarial learning techniques for security evaluation," IEEE Security & Privacy Workshops, 2018.
[47] M. Abadi and D. Andersen, "Learning-based system call exploitation," in Proc. NDSS, 2016.
[48] S. M. Bellovin and W. R. Cheswick, "Network security testing and attack automation," IEEE Internet Computing, vol. 3, no. 6, pp. 72–76, 1999.
[49] Y. Bengio and I. Goodfellow, "Deep learning applications in adversarial environments," Communications of the ACM, vol. 62, no. 2, pp. 54–63, 2019.
[50] J. Zhu and M. Zhou, "AI-based attack path prediction for enterprise systems," IEEE Access, vol. 8, pp. 190221–190233, 2020.
[51] R. Behl and S. Sahai, "Automation of ethical hacking using artificial intelligence," International Journal of Advanced Computer Science and Applications, vol. 11, no. 6, pp. 212–219, 2020.
[52] L. Allodi and F. Massacci, "Security risk assessment using vulnerability exploitability metrics," ACM Transactions on Privacy and Security, vol. 17, no. 4, pp. 1–31, 2014.
[53] A. Doupé and G. Vigna, "Scalable web application penetration testing," in Proc. ACM CCS, 2012, pp. 105–116.
[54] S. Forrest and A. Somayaji, "Anomaly detection for cyber attack identification," IEEE Symposium on Security and Privacy, 1996.
[55] J. Hoffmann and B. Nebel, "Automated planning for intelligent agents," Artificial Intelligence, vol. 76, no. 1–2, pp. 91–142, 1995.
[56] P. Trivedi and S. Kumar, "AI-driven social engineering attack simulation," Computers & Security, vol. 114, p. 102581, 2022.
[57] M. Fredrikson and S. Jha, "Model inversion attacks using machine learning," in Proc. IEEE Symposium on Security and Privacy, 2015, pp. 132–147.
[58] H. Kwon and T. Kim, "Autonomous cyber attack agents in simulated environments," Simulation Modelling Practice and Theory, vol. 103, p. 102090, 2020.

[59] S. Noel and M. Elder, "Cyber attack modeling and simulation," Journal of Defense Modeling and Simulation, vol. 10, no. 2, pp. 109–119, 2013.
[60] K. Scarfone and P. Mell, "Guide to security testing and penetration methodologies," NIST Special Publication 800-115, 2008.