The Rise and Rise of LLM-powered PenTesting Systems: Capabilities, Limitations, and the Road to Responsible Automation

Haitham S. Al-Sinani[1,2] and Chris J. Mitchell[3]

[1] Department of Cybersecurity and Quality Assurance, Diwan of Royal Court, Muscat, Oman, hsssinani@diwan.gov.om
[2] German University of Technology in Oman (GUtech), Muscat, Oman Haitham.AlSinani@gutech.edu.om
[3] Department of Information Security, Royal Holloway, University of London, Egham, UK. C.Mitchell@rhul.ac.uk

Abstract. Recent advances in large language models (LLMs) have accelerated the development of AI-enhanced penetration testing (PenTesting) systems capable of automating complex offensive security tasks. This survey reviews research published between late 2022 and early 2026, analysing a broad range of proposed systems and tools. For each system, the survey examines its objectives, architectural design, key modules, evaluation approaches, and reported limitations. By systematically comparing representative systems and identifying recurring design patterns, the paper derives a reference model for LLM-powered PenTesting and discusses evaluation practices, ethical and safety considerations, and open technical challenges. Given the rapid evolution and increasing complexity of this emerging field, this survey aims to provide researchers, practitioners, and newcomers with a clear and structured overview of the state of the art, while offering guidance for future research and development of reliable, secure, and responsible LLM-driven PenTesting systems.

1 Introduction

Ethical hacking, or Penetration testing (PenTesting) [147], is a core cybersecurity practice that enables organisations to identify and mitigate vulnerabilities before they can be exploited by adversaries. As digital infrastructures continue to expand in scale and complexity, systematic security testing has become an increasingly important component of organisational risk management. This growing importance is also reflected in market forecasts, which estimate that the global PenTesting market will grow from USD 1.92 billion in 2023 to approximately USD 6.98 billion by 2032, corresponding to a compound annual growth rate (CAGR) of 15.46% [83, 85].

The roots of PenTesting predate modern networking. In the 1960s, government and industry formed 'tiger teams' to probe their own systems for weaknesses, motivated in part by early warnings that networked computers would

create new avenues for intrusion. Anderson's 1972 report [21] subsequently formalised key steps for identifying vulnerabilities, executing attacks, and mitigating threats, while early tools such as the Security Administrator Tool for Analyzing Networks (SATAN) automated basic scanning tasks in the 1990s [74]. Despite these developments, contemporary PenTesting remains heavily reliant on skilled human operators. Although toolkits such as Kali Linux provide powerful capabilities for scanning, enumeration, traffic analysis, and exploitation, effective engagements still require testers to design attack paths, interpret intermediate feedback, and iteratively adapt their tactics to the target environment [2].

This dependence on expert judgement makes PenTesting labour-intensive, costly, and difficult to scale. Prior studies report that professional testers may spend around 80 hours on a single engagement, while empirical evidence also points to a persistent shortage of skilled practitioners [60, 135]. More broadly, the global cybersecurity workforce gap has recently been estimated at roughly 4.7–4.8 million unfilled positions, further constraining the availability of qualified testers [64,130]. At the same time, cyber threats continue to escalate in both scale and sophistication, increasing the demand for efficient and repeatable security assessment methods [64].

Earlier efforts to automate offensive security achieved only limited success. Pre-LLM approaches often modelled attack planning using deterministic attack graphs, Markov decision processes (MDPs), or partially observable MDPs. While these methods offered principled ways to reason about attack sequences, they typically assumed extensive prior knowledge of the target and produced static plans that adapted poorly to dynamic environments [135]. More generally, automation before the rise of LLMs was largely confined to reconnaissance, scanning, and other early-stage tasks, with limited ability to chain exploits, adapt to unexpected outputs, or sustain multi-step reasoning across an engagement [64].

The emergence of generative LLMs has significantly changed this landscape. Unlike earlier automation techniques, LLMs can generate commands, interpret tool outputs, reason under uncertainty, and iteratively refine their actions across multiple turns. Early studies showed that models such as GPT-3.5 could already suggest shell commands, analyse system feedback, and propose multi-step attack paths, leading to the rapid development of agentic frameworks. These advances suggest that LLMs can support, and in some settings partially automate, phases such as reconnaissance, exploitation, privilege escalation (PrivEsc), and post-exploitation.

At the same time, LLM-driven PenTesting introduces major technical and ethical concerns. These systems combine powerful reasoning capabilities with the ability to recommend or execute operationally meaningful actions, raising questions about reliability, hallucination, safety, evaluation, privacy, and real-world applicability. Ethical analyses of recent LLM-enabled offensive-security prototypes further highlight the dual-use dilemma: tools intended to strengthen defence may also lower the barrier to offensive misuse if released without appropriate safeguards [63]. As a result, any serious treatment of LLM-powered

PenTesting must consider not only capability and performance, but also governance, disclosure, and responsible deployment.

This survey examines the evolving landscape of LLM-powered PenTesting systems published between late 2022 and early 2026. Drawing on peer-reviewed literature, arXiv preprints, technical reports, and relevant standards, we analyse representative systems in terms of their architectures, core components, operational capabilities, and evaluation methodologies. We also compare these systems across key dimensions, derive a reference model capturing common design patterns, and discuss benchmarks, ethical considerations, limitations, open challenges, and promising future research directions. In doing so, the survey aims to provide both researchers and practitioners with a clear and structured overview of the state of the art.

The remainder of this paper is organised as follows. Section 2 reviews the ethical hacking landscape. Section 3 provides background on LLMs. Section 4 discusses key techniques and design patterns relevant to LLM-driven PenTesting systems. Section 5 surveys representative systems. Section 6 compares them, and Section 7 presents their classification. Section 8 discusses limitations and challenges, Section 9 reviews related work, and Section 10 concludes the paper.

For ease of reference, Table 1 summarises the key acronyms used throughout this survey.

2 The Ethical Hacking Landscape

2.1 Introduction

Ethical hacking, or PenTesting [18, 26, 45, 133, 141, 147, 164], involves applying hacking techniques to help organisations strengthen their security posture. It is a discipline that requires a blend of technical expertise, creativity, and a deep understanding of potential threats. Unlike illegal hacking, ethical hacking is the authorised practice of bypassing system security to identify potential vulnerabilities and thereby help protect against real cyber attacks. Ethical hacking typically follows a structured approach involving five phases, as follows.

1. **Reconnaissance:** This involves gathering detailed information about the target system, including public data, network structures, IP address ranges, live hosts, and system configurations. The objective is to understand the target's environment and identify potential vulnerabilities.
2. **Scanning and Enumeration:** During this phase, ethical hackers first perform network scanning to identify active hosts and reachable systems within the target environment. This is followed by enumeration, where more detailed information is gathered about the discovered systems, including open ports, running services, software versions, and system configurations. The objective is to build a detailed profile of the target infrastructure and identify services or weaknesses that may provide a viable path for gaining initial access in the subsequent attack stage.

3. **Gaining Access:** During this stage, hackers leverage identified vulnerabilities to penetrate the target system. The aim is to gain an initial foothold, demonstrating how an attacker could potentially breach security measures and obtain unauthorised access.
4. **Maintaining and Elevating Access:** This stage involves researching and developing ways to gain elevated access and re-enter the system undetected. Techniques such as backdoors are implemented to enable long-term, potentially privileged access. Additionally, tactics like pivoting and lateral movement are employed to navigate through the network, accessing various systems to increase control and maintain persistence within the environment.
5. **Covering Tracks and Reporting:** This stage focuses on erasing any traces of the hacking activity to avoid detection and restoring the target system to its original state. Following this, ethical hackers prepare a comprehensive report detailing their findings and providing recommendations for improving the system's security posture.

2.2 PenTesting Standards

PenTesting [1,18,23,26,49,50,133,138,141,144,149,160,164] methodologies[4] provide structured approaches to simulate adversarial behaviour, identify vulnerabilities, and assess organisational security posture. Multiple industry-recognised standards and frameworks offer guidance, each with its own scope and emphasis. This subsection surveys some of the most widely adopted PenTesting standards and frameworks. Table 2 provides a comparative overview of the principal approaches discussed, summarising their originating organisations, structural characteristics, primary focus, and typical application contexts.

2.2.1 NIST SP 800-115

NIST SP 800-115 [147] is a U.S. government standard that defines a structured methodology for security assessments, including PenTesting. NIST SP 800-115 divides the process into four major phases:

1. **Planning** — Define scope, objectives, rules of engagement, and obtain management authorisation.
2. **Discovery** — Collect information and perform vulnerability identification using both passive and active techniques.
3. **Attack** — Exploit identified vulnerabilities to validate their impact and determine possible compromise paths.
4. **Reporting** — Document findings, provide risk ratings, and recommend mitigation strategies.

[4] https://owasp.org/www-project-web-security-testing-guide/v41/3-The_OWASP_Testing_Framework/1-Penetration_Testing_Methodologies

Unlike some practitioner frameworks, NIST emphasises formal risk management integration and compliance, making it particularly suitable for regulated environments. Its phased approach aligns closely with enterprise governance processes and legal considerations.

2.2.2 PenTesting Execution Standard (PTES)

The PTES [122] provides a detailed technical and procedural blueprint for conducting penetration tests. It defines seven stages, as follows.

1. **Pre-engagement Interactions** — Establish scope, objectives, and communication protocols.
2. **Intelligence Gathering** — Collect information about the target through OSINT (Open-Source Intelligence), network mapping, and enumeration.
3. **Threat Modelling** — Translate gathered data into threat scenarios by understanding attack surfaces and business context.
4. **Vulnerability Analysis** — Identify weaknesses through manual analysis and automated scanning.
5. **Exploitation** — Leverage vulnerabilities to gain access and demonstrate potential impact.
6. **Post-Exploitation** — Assess the value of the compromised environment, perform lateral movement, and determine business impact.
7. **Reporting** — Deliver comprehensive documentation and remediation guidance.

PTES is practitioner-focused and technology-agnostic, making it widely adopted in professional red teaming and PenTesting engagements. Its explicit inclusion of threat modelling differentiates it from more linear models.

2.2.3 Open Source Security Testing Methodology Manual (OSSTMM)

The OSSTMM [69], maintained by the Institute for Security and Open Methodologies (ISECOM), is a peer-reviewed methodology emphasising measurable and repeatable testing. OSSTMM focuses on *channels* (communications, physical, human) and *operational security metrics*, prioritising quantifiable results over narrative reports. Key aspects include:

- **Verification vs. Validation:** Ensuring that testing measures actual security controls rather than assumed policies.
- **Trust Analysis:** Evaluating how trust relationships affect attack surfaces.
- **Rules of Engagement:** Strictly defined to ensure legal and ethical compliance.

OSSTMM is particularly useful for organisations seeking audit-grade, metrics-driven assessments across multiple domains (network, physical, human).

2.2.4 Information Systems Security Assessment Framework (ISSAF)

ISSAF [105], developed by the Open Information Systems Security Group, provides a comprehensive framework covering both technical and managerial aspects of security assessments. It divides the process into 4 key stages, as follows.

1. **Planning and Preparation** — Define objectives, resources, and scope.
2. **Assessment** — Conduct vulnerability assessment, PenTesting, and risk analysis.
3. **Post-Assessment** — Correlate findings, assess impact, and develop remediation strategies.
4. **Reporting** — Present findings to stakeholders.

ISSAF is less frequently updated compared to NIST or PTES but remains a reference point for multi-layered assessment methodologies, especially in academic and governmental contexts.

2.2.5 Cyber Kill Chain (CKC)

Introduced by Lockheed Martin's Intelligence Driven Defense methodology, the CKC [73, 121] is a linear framework that decomposes a targeted cyberattack into seven stages. By mapping an intrusion to these stages, defenders can detect and disrupt adversary operations early in their lifecycle. The seven stages are as follows.

1. **Reconnaissance.** Attackers survey their target to identify weaknesses. This includes gathering publicly available information (OSINT), scanning for exposed services and deploying network probes to map the environment.
2. **Weaponization.** Using the intelligence collected, adversaries develop or assemble a malicious payload (virus, worm or Trojan) paired with an exploit that will leverage a specific vulnerability in the target system.
3. **Delivery.** The weaponized payload is transmitted to the victim via a chosen vector such as phishing emails, drive-by downloads or compromised hardware/software supply chains.
4. **Exploitation.** Once delivered, the exploit triggers and executes the payload by exploiting the vulnerability on the victim system, thereby establishing a foothold. This stage corresponds to the "break in" moment of the attack.
5. **Installation.** The attacker installs additional malware to persist within the environment. Persistence mechanisms include backdoors, remote shells or PrivEsc tools.
6. **Command and Control (C2).** With persistence established, the attacker opens a C2 channel to communicate with the compromised system and issue further instructions, often using obfuscation to evade detection.
7. **Actions on Objectives.** Finally, the attacker accomplishes their primary goal—be it data exfiltration, system destruction, encryption for ransom or lateral movement to other targets.

By dissecting intrusions into these phases, the CKC emphasises that breaking any single link can thwart the entire attack. However, subsequent research highlights several limitations: the model assumes a linear progression and is heavily focused on malware delivered at the network perimeter; case studies show that phases can be bypassed or occur in a different order and that additional activities such as PrivEsc and credential access can occur after C2 is established [120]. Consequently, contemporary defenders often pair the kill chain with richer models such as MITRE ATT&CK to capture non-linear adversary behaviour and to expand the final stage into multiple tactics (PrivEsc, lateral movement, exfiltration, impact) [120].

Note that attack methodologies such as NIST SP 800-115 and the CKC describe high-level phases of an attack rather than specific techniques or exploits [101]. For example, the CKC conceptualises an intrusion as a sequence of stages — Reconnaissance, Weaponisation, Delivery,etc. — providing a structured framework for analysing adversary behaviour rather than prescribing concrete attack steps.

2.2.6 Attack Life Cycle (ALC)

The ALC [92,93] (see Fig. 1) is a threat model developed by Mandiant to describe the end-to-end sequence of actions typically observed during targeted cyber intrusions. Unlike the CKC, which focuses on early-phase perimeter breaches, the ALC emphasises the full operational scope of threat actors, particularly in advanced persistent threat (APT) campaigns. The model highlights how attackers maintain control and expand their reach within the environment after the initial compromise. The ALC's eight phases are as follows.

1. **Initial Reconnaissance.** Adversaries gather intelligence on the target organisation, including personnel, systems, applications, and external-facing assets. This stage may involve both passive OSINT and active probing.
2. **Initial Compromise.** Attackers gain an initial foothold using methods such as phishing, web exploits, or malicious documents. This mirrors the delivery and exploitation stages in the CKC.
3. **Establish Foothold.** Once inside, adversaries install malware or tools (e.g., droppers, backdoors) to maintain persistent access. Initial payloads often serve as staging points for additional capabilities.
4. **Escalate Privileges.** Attackers attempt to elevate their access rights by exploiting misconfigurations, credential reuse, or unpatched vulnerabilities. This enables broader control over the system or domain.
5. **Internal Reconnaissance.** With higher privileges, attackers map the internal network, locate critical systems, and identify potential lateral movement paths or data of interest.
6. **Move Laterally.** Using stolen credentials or exploits, adversaries pivot to other hosts and accounts within the environment, extending their access to meet operational objectives.

7. **Maintain Presence.** Advanced attackers establish redundant access channels and deploy additional tools to ensure long-term persistence, even if initial footholds are discovered and removed.
8. **Complete Mission.** The final stage involves exfiltration, destruction, or manipulation of data, depending on the attacker's objectives. This could also include disrupting operations, establishing ransomware encryption, or preparing for a future attack.

The ALC provides a more holistic view of attacker behaviour after breaching the perimeter, accounting for post-exploitation manoeuvres such as PrivEsc, lateral movement, and long-term persistence. It is widely used in incident response, threat intelligence, and purple teaming exercises to understand APT-level adversaries. As with other models, it is often used in conjunction with MITRE ATT&CK to achieve finer-grained mapping of tactics and techniques.

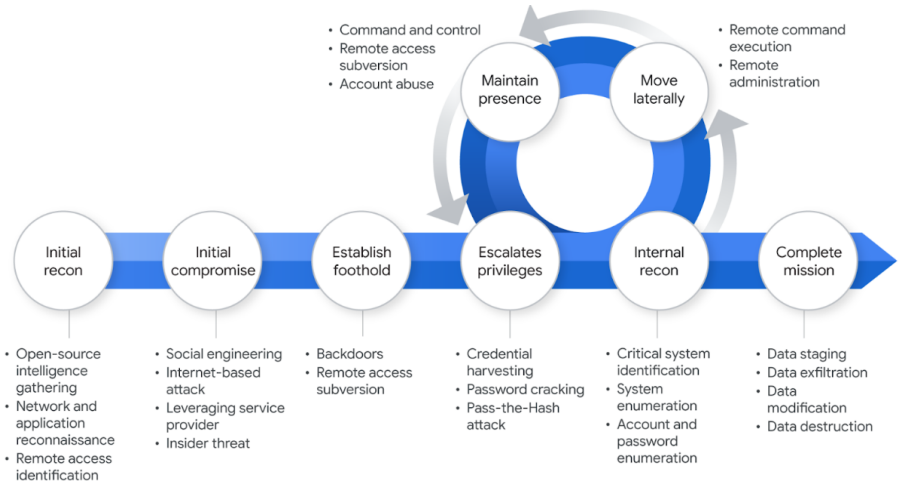


Fig. 1: Mandiant's Attack Life Cycle (reproduced from [93]).

2.2.7 MITRE ATT&CK

MITRE ATT&CK [146] is a structured knowledge base that catalogues adversary behaviour, particularly those associated with advanced persistent threats (APTs). It organises this information using a hierarchical model commonly referred to as "TTPs" — Tactics, Techniques and Procedures.

- **Tactics** represent the high-level objectives an attacker aims to achieve during an operation, such as reconnaissance, PrivEsc or data theft.
- **Techniques** describe specific ways to accomplish a given tactic. For example, *Abuse Elevation Control Mechanism: Sudo and Sudo Caching*[5] illus-

[5] `https://attack.mitre.org/techniques/T1548/003/`

trates a method for PrivEsc, while *Steal or Forge Kerberos Tickets: Kerberoasting*[6] exemplifies credential theft.
– **Procedures** provide the granular, real-world implementation details of a technique, documenting exactly how adversaries execute it in practice.

The ATT&CK matrix for enterprise environments includes the following fourteen tactics (see Fig. 2), ordered to reflect the typical progression of an adversary through the attack lifecycle:

1. **Reconnaissance** — Gathering preliminary information about the target from external sources (e.g., OSINT, scanning).
2. **Resource Development** — Acquiring infrastructure, credentials, or tools required for the attack (e.g., domains, malware, compromised accounts).
3. **Initial Access** — Gaining entry into the target environment (e.g., spear phishing, exploiting public-facing applications).
4. **Execution** — Running malicious code within the target environment (e.g., command-line execution, script interpreters).
5. **Persistence** — Establishing long-term footholds to survive reboots or credential changes (e.g., autostart execution, account manipulation).
6. **Privilege Escalation (PrivEsc)** — Gaining higher-level permissions on compromised systems (e.g., exploiting system processes, abusing sudo).
7. **Defense Evasion** — Avoiding detection and evading defensive measures (e.g., obfuscating commands, disabling logging).
8. **Credential Access** — Obtaining credentials through dumping, brute force, or phishing (e.g., credential harvesting, keylogging).
9. **Discovery** — Mapping the internal network and identifying targets (e.g., service discovery, system enumeration).
10. **Lateral Movement** — Moving between systems in the network to expand access (e.g., remote service exploitation, session hijacking).
11. **Collection** — Gathering sensitive data for exfiltration (e.g., clipboard capture, file collection).
12. **Command and Control (C2)** — Establishing communication channels with compromised systems (e.g., reverse shells, encrypted tunnels).
13. **Exfiltration** — Transferring stolen data out of the organisation (e.g., over web protocols, removable media).
14. **Impact** — Performing destructive or disruptive actions to achieve attacker objectives (e.g., ransomware, data wiping, service interruption).

By capturing adversary behaviour in a consistent, research-driven framework, MITRE ATT&CK has become a cornerstone reference for threat modelling, detection engineering and red-team operations.

MITRE Caldera [14] is an open-source cyber operations automation platform primarily used in Purple-Teaming exercises, where offensive and defensive teams collaborate to evaluate detection and response capabilities in a controlled setting. In such exercises, the attacker emulates the TTPs of a known APT

[6] `https://attack.mitre.org/techniques/T1558/003/`

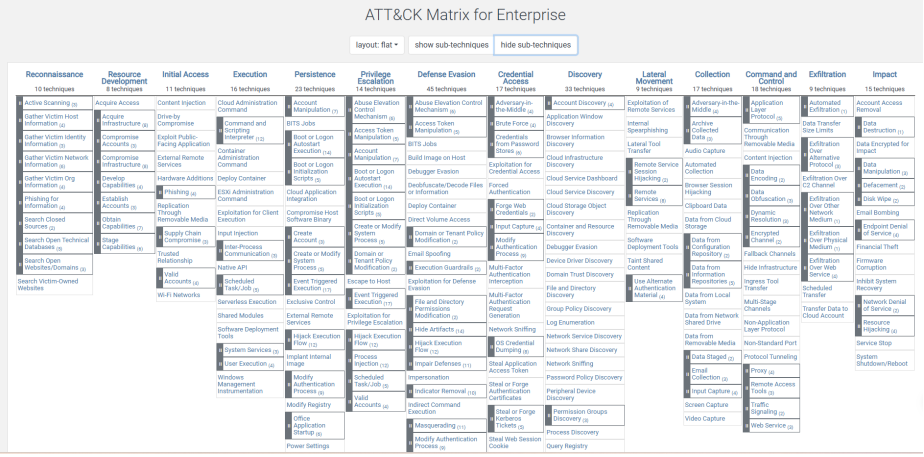


Fig. 2: Att&ck Matrix for Enterprise [97]

group, executes predefined attack steps, and jointly analyses whether the defenders were able to detect the activity and apply appropriate countermeasures. Caldera supports this workflow by allowing operators to configure and automatically execute a fixed set of attack techniques — such as SMB enumeration or password spraying using static credential lists — mapped to the MITRE ATT&CK framework. While this automation resembles vulnerability scanning at a superficial level, Caldera operates strictly within Assumed Breach scenarios and does not perform autonomous strategy generation or adaptive planning. This limitation is intentional: the high-level attack strategy is manually specified to faithfully reproduce documented APT behaviour, rather than to discover new attack paths or optimise exploitation sequences. As a result, Caldera serves as a controlled emulation and validation tool rather than a general-purpose or autonomous PenTesting system.

2.2.8 OWASP Top 10

The OWASP Top 10 [113] is a community-driven standard awareness document highlighting the ten most critical web application security risks. Updated every three to four years, it serves as a baseline for secure coding, testing, and risk management.

The latest edition, OWASP Top 10:2025[7], revises and reorganises several categories to reflect emerging threats such as supply-chain attacks and improved monitoring practices. The ten risk categories identified in the 2025 edition are as follows.

1. A01:2025 – Broken Access Control

[7] https://owasp.org/Top10/2025/

2. A02:2025 – Security Misconfiguration
3. A03:2025 – Software Supply Chain Failures
4. A04:2025 – Cryptographic Failures
5. A05:2025 – Injection
6. A06:2025 – Insecure Design
7. A07:2025 – Authentication Failures
8. A08:2025 – Software or Data Integrity Failures
9. A09:2025 – Security Logging and Alerting Failures
10. A10:2025 – Mishandling of Exceptional Conditions

While not a PenTesting methodology per se, the OWASP Top 10 provides a critical reference for web-focused assessments, ensuring coverage of the most prevalent vulnerabilities observed in real-world applications.

2.2.9 PCI PenTesting Guide

The Payment Card Industry Data Security Standard (PCI DSS [117]) defines mandatory PenTesting requirements under Requirement 11.3. The *PCI DSS PenTesting Guidance* provides best practices and minimum expectations for testing cardholder data environments (CDE). Key elements are as follows.

- **PenTesting Components** — Covering both network-layer and application-layer testing.
- **Qualifications of a Penetration Tester** — Ensuring independence and technical competence.
- **Methodology** — Based on industry-accepted approaches with internal and external coverage.
- **Scope Validation** — Testing to verify segmentation and scope reduction.
- **Reporting Guidelines** — Documenting findings, impact, and remediation.

PCI DSS emphasises the protection of cardholder data, making this guide critical for financial institutions and merchants handling payment information.

2.2.10 Penetration Testing Framework (PTF)

The PTF is a comprehensive, hands-on methodology providing step-by-step guidance and recommended tools for each testing category [123]. It covers a wide range of areas, including:

- Network Footprinting and Reconnaissance;
- Discovery & Probing;
- Enumeration and Password Cracking;
- Vulnerability Assessment;
- Server and Network Backbone Testing;
- Wireless and VoIP Security;
- Physical Security;
- Specialised Testing (AS/400, Cisco, Citrix, Bluetooth); and
- Reporting Templates and Deliverables.

PTF's extensive technical focus and tool integration make it popular for operational security teams seeking practical, repeatable testing procedures.

2.3 PenTesting Tools & Resources

2.3.1 Overview

Table 3 summarises representative PenTesting tools and frameworks discussed throughout this survey, grouped by functional category and illustrating their primary roles within the PenTesting lifecycle.

2.3.2 Traditional PenTesting Tools

Before the emergence of AI-driven systems, PenTesting workflows relied heavily on a core set of specialised tools designed to support reconnaissance, vulnerability assessment, exploitation, and post-exploitation activities. These traditional toolkits remain essential today and often serve as the underlying components orchestrated by modern automated or AI-augmented frameworks.

Kali Linux[8] is the de facto standard Linux distribution for PenTesting, bundling hundreds of security-focused utilities in a single, ready-to-use platform. Its curated toolset spans the entire testing lifecycle, from reconnaissance to post-exploitation (see Fig. 3), and is widely adopted in both professional engagements and academic training.

For network scanning and service enumeration, nmap[9] is *the* cornerstone tool. It enables testers to discover live hosts, map open ports, and identify running services (see Table 4), forming the basis for subsequent vulnerability analysis. Complementing `nmap`, vulnerability scanners such as OpenVAS and Tenable Nessus (see Section 2.3.4) provide automated assessment of discovered nodes.

Packet capture and traffic analysis are typically conducted using Wireshark[10], a widely used network protocol analyser capable of inspecting live traffic or saved captures at a granular level. Wireshark assists in identifying misconfigurations, detecting potential MITM opportunities, and validating exploit success.

Web application security testing commonly relies on interception proxies such as Burp Suite [91][11], which enables testers to capture, inspect and modify HTTP(S) traffic between the client and server in real time. Acting as a MITM proxy, Burp Suite facilitates tasks such as parameter tampering, session analysis and automated vulnerability scanning. Alternatives and complementary tools include OWASP ZAP[12], which provides similar interception and scanning capabilities in an open-source package. These tools are critical in assessing web application logic and implementing targeted exploit chains, bridging automated scanning with manual testing.

Credential attacks form another core aspect of PenTesting. Hydra[13] is widely used for online brute-force attacks against network services, supporting numerous protocols including SSH, FTP and HTTP. For offline password auditing and

[8] `https://www.kali.org/`
[9] `https://nmap.org/`
[10] `https://www.wireshark.org/`
[11] `https://portswigger.net/burp`
[12] `https://www.zaproxy.org/`
[13] `https://github.com/vanhauser-thc/thc-hydra`

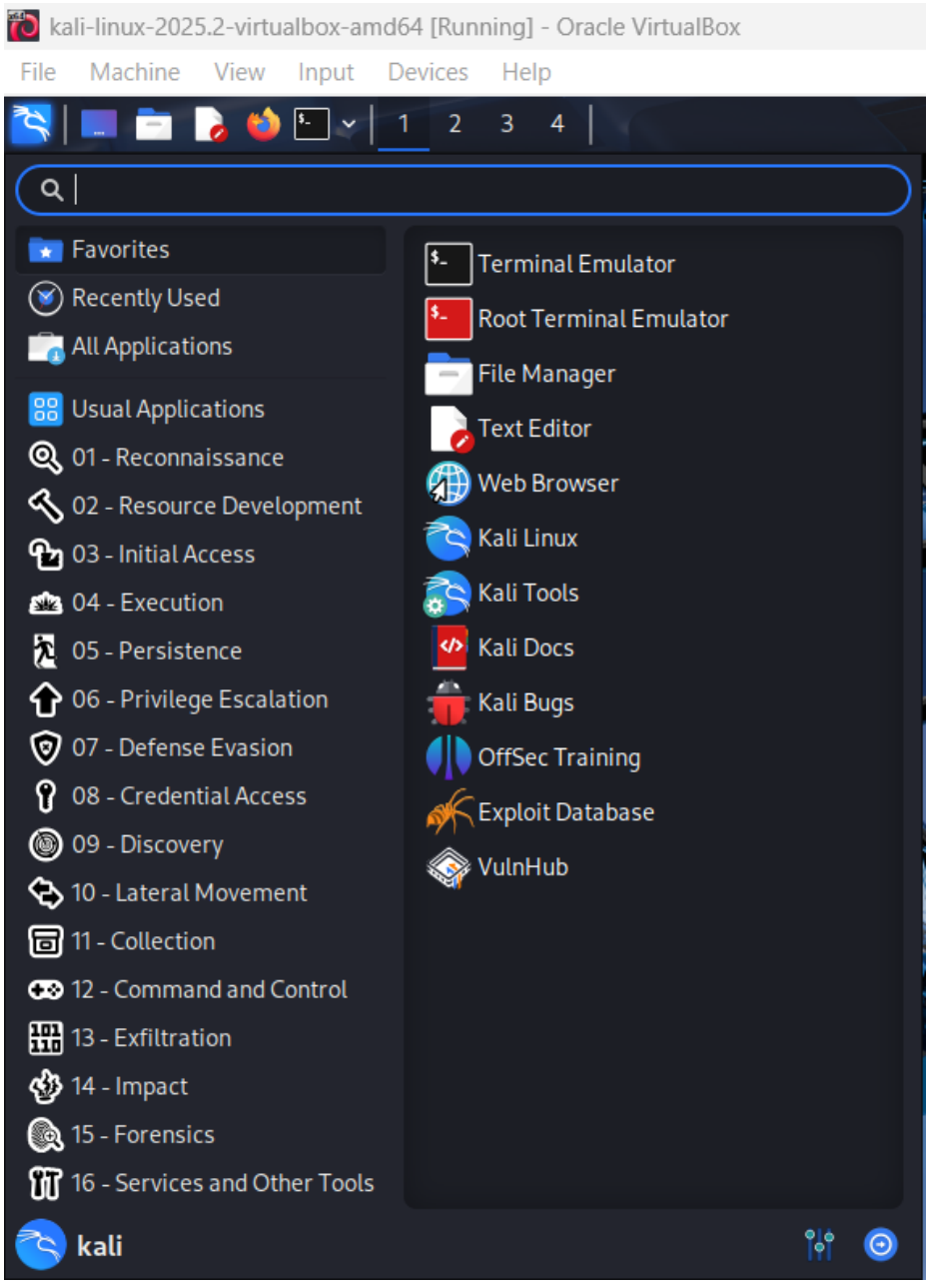


Fig. 3: Kali Linux

hash cracking, John the Ripper[14] and Hashcat[15] are industry-standard tools capable of performing dictionary, brute-force and rule-based attacks at scale. Together, these utilities enable testers to evaluate password strength and credential management practices across both network services and stored authentication data.

These traditional tools provide the foundation upon which modern GenAI-augmented PenTesting systems build. Many LLM-powered agents integrate their outputs, automate command generation, or leverage them as execution backends, highlighting their continued relevance in both manual and automated testing contexts.

2.3.3 PenTesting Frameworks

A number of dedicated frameworks have emerged to streamline and standardise offensive security workflows. The Metasploit Framework[16] is one of the most widely adopted, providing a modular platform for vulnerability discovery, exploitation and post-exploitation. Metasploit's extensive exploit, payload, and auxiliary module libraries make it a de facto standard in both professional and academic PenTesting. Other frameworks such as Core Impact[17] and Immunity Canvas[18] offer similar capabilities in commercial form, while toolkits like Kali Linux[19] integrate Metasploit alongside reconnaissance and enumeration utilities such as Nmap and Wireshark. These frameworks serve as a practical baseline for modern engagements and are frequently integrated into AI-augmented systems, providing the underlying exploitation and payload delivery mechanisms that LLM-driven agents can orchestrate.

2.3.4 Vulnerability Scanners

Automated vulnerability scanners form a cornerstone of the discovery phase by systematically identifying weaknesses before manual exploitation. Broadly, they can be divided into *network and host vulnerability scanners*, which focus on network-connected devices, infrastructure services and operating systems (OSs), and *web application scanners*, which target application-layer flaws.

Network Scanners. Among network and host scanners, OpenVAS[20], part of the Greenbone Vulnerability Management (GVM) suite, is a leading open-source platform for network-based assessments. Tenable Nessus[21] is one of the most widely used commercial tools, offering extensive plugin coverage

[14] https://www.openwall.com/john/
[15] https://hashcat.net/hashcat/
[16] https://www.metasploit.com/
[17] https://www.coresecurity.com/core-impact
[18] https://immunityinc.com/
[19] https://www.kali.org/
[20] https://www.openvas.org/
[21] https://www.tenable.com/products/nessus

and compliance checks. Enterprise solutions such as Qualys Vulnerability Management[22] and Rapid7 Nexpose/InsightVM[23] add features like continuous asset discovery, risk prioritisation and integration with security information and event management (SIEM) systems, scanning a wide range of nodes including servers, endpoints, network appliances and virtualised assets.

Web Scanners. Web vulnerability scanners include a broad ecosystem of both open-source and commercial tools designed to detect application-layer flaws such as injection attacks, XSS (XSS), insecure authentication, and misconfigurations. Acunetix[24] and Invicti (formerly Netsparker)[25] are leading commercial solutions with strong automation and reporting capabilities. Open-source options such as Nikto[26], Arachni[27], W3af[28] (Web Application Attack and Audit Framework), Skipfish[29], and IronWASP[30] offer flexible scanning engines suited for research and academic use. Widely adopted tools like the OWASP Zed Attack Proxy (ZAP)[31] and Burp Suite[32] integrate automated scanning with powerful interception and manual testing capabilities, making them indispensable in professional PenTesting workflows. Additional utilities such as Wfuzz[33], Gobuster[34], Commix[35], and XSStrike[36] provide targeted fuzzing and injection testing for parameters, while commercial enterprise platforms like Qualys Web Application Scanning (WAS)[37] and IBM AppScan[38] support large-scale continuous assessment with integration into CI/CD pipelines.

Collectively, these tools provide coverage across rapid reconnaissance, vulnerability enumeration, and advanced application-layer security testing, forming a critical part of modern vulnerability management and PenTesting practices.

In addition to its well-known exploitation capabilities, the Metasploit Framework [80][39] also includes modules for vulnerability scanning. These can help identify known weaknesses in target systems prior to exploitation, complementing

[22] https://www.qualys.com/
[23] https://www.rapid7.com/products/insightvm/
[24] https://www.acunetix.com/
[25] https://www.invicti.com/
[26] https://cirt.net/Nikto2
[27] https://www.arachni-scanner.com/
[28] http://w3af.org/
[29] https://code.google.com/archive/p/skipfish/
[30] https://ironwasp.org/
[31] https://www.zaproxy.org/
[32] https://portswigger.net/burp
[33] https://github.com/xmendez/wfuzz
[34] https://github.com/OJ/gobuster
[35] https://commixproject.com/
[36] https://github.com/s0md3v/XSStrike
[37] https://www.qualys.com/apps/web-app-scanning/
[38] https://www.ibm.com/security/application-security/appscan
[39] https://www.metasploit.com/

dedicated scanning tools by integrating detection and exploitation into a unified workflow.

Modern penetration tests often combine both categories: network and host scanners for infrastructure and device-level weaknesses, and web application scanners for application-layer flaws. While these tools excel at breadth and repeatability, they typically stop short of active exploitation, making them complementary to frameworks like Metasploit. Recent LLM-powered PenTesting systems increasingly integrate outputs from both network and web scanners as structured input, enabling AI agents to chain discovered vulnerabilities into complete exploit paths.

2.3.5 Privilege Escalation (PrivEsc) Hacking Tools

PrivEsc [84] is a critical stage of PenTesting that occurs after initial access has been gained, allowing a tester to move from a limited user context to higher-level administrative or `root` privileges. A range of dedicated tools and scripts have been developed to automate local enumeration, identify misconfigurations, and suggest exploitation paths for both Linux and Windows systems.

Linux. On Linux platforms, `LinPEAS`[40] (Linux Privilege Escalation Awesome Script) is one of the most widely used enumeration utilities, performing comprehensive checks for kernel exploits, SUID binaries, misconfigured services and weak file permissions. `Linux Exploit Suggester`[41] provides a database-driven approach, matching kernel and configuration data to known local exploits. Tools such as `pspy`[42] enable process monitoring without elevated privileges, helping testers detect scheduled tasks and scripts that can be hijacked for PrivEsc.

Windows. For Windows environments, `WinPEAS`[43] (Windows Privilege Escalation Awesome Script) automates the discovery of misconfigurations, weak ACLs, registry permissions and privilege assignments. PowerShell-based frameworks such as PowerUp[44] and SharpUp[45] focus on PrivEsc via Windows-specific mechanisms, including service misconfigurations, DLL hijacking and token manipulation.

Complementary to enumeration tools are exploit frameworks and databases (see section 2.3.3). `GTFOBins`[46] collects Unix binaries that can be abused for PrivEsc when misconfigured with elevated capabilities, while `LOLBAS`[47] provides

[40] `https://github.com/carlospolop/PEASS-ng/tree/master/linPEAS`
[41] `https://github.com/The-Z-Labs/linux-exploit-suggester`
[42] `https://github.com/DominicBreuker/pspy`
[43] `https://github.com/carlospolop/PEASS-ng/tree/master/winPEAS`
[44] `https://github.com/PowerShellEmpire/PowerTools/tree/master/PowerUp`
[45] `https://github.com/GhostPack/SharpUp`
[46] `https://gtfobins.github.io/`
[47] `https://lolbas-project.github.io/`

a similar resource for Windows binaries and scripts. These are frequently used in combination with manual testing to convert misconfigurations into working exploits.

Modern AI-driven PenTesting agents increasingly integrate PrivEsc tools into automated workflows, using their outputs as structured input for reasoning modules that chain vulnerabilities into escalation paths. As PrivEsc often determines the success of an entire engagement, these tools remain a cornerstone of both manual and AI-augmented post-exploitation testing.

2.3.6 Wireless Hacking Tools

Wireless network security assessment forms a critical component of many PenTesting engagements, targeting weaknesses in Wi-Fi configurations, encryption protocols, and access controls. A range of specialised tools has been developed to support reconnaissance, packet capture, key recovery and attack simulation in wireless environments.

One of the most widely used toolsets is the Aircrack-ng suite[48], which provides a comprehensive collection of utilities for monitoring, capturing and attacking IEEE 802.11 wireless networks. Components such as `airodump-ng` enable passive network discovery and packet capture, while `aireplay-ng` supports packet injection for replay and deauthentication attacks. The core `aircrack-ng` utility performs key recovery through WEP and WPA/WPA2-PSK cracking using captured handshakes. Complementary tools such as Kismet[49] offer passive wireless network detection and intrusion detection system (IDS) functionality, mapping access points and clients without active probing.

For targeted attacks, Reaver[50] exploits weaknesses in Wi-Fi Protected Setup (WPS) implementations to recover WPA/WPA2 passphrases, while Wifite[51] automates the process of scanning, capturing and cracking common Wi-Fi configurations. Tools such as Fern WiFi Cracker[52] provide graphical interfaces for automating wireless attacks, making them accessible for both professional and educational use.

Bluetooth security testing complements Wi-Fi assessments in wireless engagements. Utilities like BlueMaho[53] and BlueHydra[54] perform Bluetooth reconnaissance, device enumeration and vulnerability scanning, helping testers evaluate short-range wireless security implementations.

Together, these tools form the core arsenal for wireless PenTesting. They are often bundled in dedicated distributions such as Kali Linux, where they integrate with general-purpose frameworks and reporting workflows. Modern

[48] `https://www.aircrack-ng.org/`
[49] `https://www.kismetwireless.net/`
[50] `https://code.google.com/archive/p/reaver-wps/`
[51] `https://github.com/derv82/wifite2`
[52] `https://github.com/savio-code/fern-wifi-cracker`
[53] `https://sourceforge.net/projects/bluemaho/`
[54] `https://github.com/greatscottgadgets/blue_hydra`

AI-augmented PenTesting systems can leverage their outputs for automated analysis of wireless environments, chaining handshake captures and encryption weaknesses into full exploit paths.

2.3.7 Emerging IoT PenTesting Tools

With the rapid proliferation of Internet of Things (IoT) devices, PenTesting has expanded beyond traditional IT networks to encompass embedded systems, home automation platforms, industrial control systems (ICS) and smart sensors. IoT security assessments must account for constrained devices, proprietary protocols and diverse wireless standards such as ZigBee, Z-Wave and LoRaWAN.

Specialised toolkits are emerging to address these challenges. KillerBee[55] targets ZigBee networks, providing capabilities for packet capture, frame injection and key extraction in common IoT environments such as smart lighting and industrial wireless sensors. Z-Wave auditing is supported by frameworks such as Z-Wave Sniffer[56], which captures and decodes traffic from home automation networks. Tools like IoT Inspector[57] enable passive network traffic analysis of consumer IoT devices, identifying insecure communications and firmware update mechanisms.

As IoT ecosystems frequently integrate Bluetooth Low Energy (BLE), utilities such as Gatttool[58] and BtleJack[59] provide packet sniffing and MITM capabilities for BLE devices. Combined with hardware debuggers (e.g., JTAG, SWD) and firmware extraction frameworks, these tools form the basis for modern IoT PenTesting workflows.

Emerging AI-driven systems are beginning to incorporate IoT-specific modules that automate device fingerprinting, protocol fuzzing and vulnerability correlation, highlighting the growing need for intelligent and scalable approaches to IoT security assessment.

Table 5 summarises representative wireless security tools commonly used during different stages of the PenTesting process, highlighting how specialised utilities support activities ranging from reconnaissance and access acquisition to post-compromise validation and wireless/IoT device assessment.

2.3.8 Insecure Web Apps for Training

To provide a safe and legal environment for practising offensive and defensive security techniques, several deliberately vulnerable web applications have been created for use in PenTesting training and Capture-the-Flag (CTF) competitions. These platforms replicate common web application vulnerabilities and misconfigurations, allowing testers to develop practical skills without targeting

[55] https://github.com/riverloopsec/killerbee
[56] https://www.silabs.com/developers/z-wave-sniffer
[57] https://iot-inspector.com/
[58] https://manpages.debian.org/unstable/bluez/gatttool.1.en.html
[59] https://github.com/virtualabs/btlejack

production systems. Notable examples of deliberately insecure web applications designed for PenTesting training and education include the following.

1. **OWASP Juice Shop.**[60] One of the most popular deliberately insecure web applications, Juice Shop is designed to cover the entire OWASP Top 10. It intentionally includes vulnerabilities such as SQL injection, cross-site scripting (XSS), insecure direct object references (IDOR), broken authentication, and sensitive data exposure. Juice Shop is widely used in training programmes, university courses and online CTFs due to its gamified challenge-based structure, extensive documentation, and modern technology stack (Node.js and Angular).
2. **Damn Vulnerable Web Application (DVWA).**[61] DVWA is a PHP and MySQL-based training environment focused on web exploitation techniques. It offers configurable security levels, allowing learners to practise attacks such as SQL injection, XSS, command execution, and cross-site request forgery (CSRF) at varying levels of difficulty. DVWA is especially useful for demonstrating how small configuration changes can significantly alter an application's security posture.
3. **Mutillidae II.**[62] Mutillidae II is an OWASP Top 10 training platform designed for both offensive and defensive web security education. It provides over 40 different vulnerabilities and is often deployed in virtual labs for testing tools like Burp Suite and Nikto. Mutillidae also integrates defensive lessons, making it valuable for teaching secure coding practices alongside exploitation.
4. **OWASP WebGoat.**[63] WebGoat is a lesson-based OWASP project aimed at teaching web application security concepts. Each lesson introduces a specific vulnerability, guides the user through exploiting it, and explains secure remediation techniques. Its structured approach makes it suitable for formal coursework and self-paced learning.
5. **bWAPP (buggy Web Application).**[64] bWAPP is a flexible vulnerable web application covering over 100 security issues. It supports both low- and high-complexity scenarios and is designed to work with tools such as OWASP ZAP and Burp Suite. Its wide range of vulnerabilities makes it ideal for comprehensive web security training.

These intentionally vulnerable platforms are integral to cybersecurity education and professional development. They provide realistic and repeatable environments for testing manual techniques and automated tools alike. Increasingly, they are also used to benchmark the performance of AI-driven PenTesting agents against real-world web application vulnerabilities.

[60] https://owasp.org/www-project-juice-shop/
[61] http://www.dvwa.co.uk/
[62] https://github.com/webpwnized/mutillidae
[63] https://owasp.org/www-project-webgoat/
[64] http://www.itsecgames.com/

2.3.9 Network Topology Emulation for PenTesting

To simulate realistic attack and defence scenarios, PenTesters often rely on network topology emulation environments that replicate enterprise infrastructures in a controlled and repeatable manner. These environments allow the safe deployment of vulnerable machines, security appliances, and attacker platforms, enabling end-to-end testing of reconnaissance, exploitation and post-exploitation workflows.

- **Cyber Ranges.** are comprehensive platforms designed to create full-scale virtual infrastructures for cybersecurity training, red/blue team exercises, and defensive monitoring. The KYPO Cyber Range Platform (KYPO CRP)[65] is a fully open-source solution developed by Masaryk University since 2013 and released under the MIT license in 2020. It is widely adopted in academic curricula and national cyber defence exercises. It supports scenario-based training, scalable topologies via OpenStack, and real-time learner progress tracking, making it ideal for immersive PenTesting and red team simulations.
- **GNS3 (Graphical Network Simulator 3).**[66] is one of the most popular open-source platforms for designing and emulating network topologies. It allows testers to integrate real network OS images (e.g., Cisco IOS, Juniper) alongside virtual machines (VMs) running Kali Linux, Metasploitable, and other targets, making it suitable for both PenTesting and defensive network validation.
- **EVE-NG (Emulated Virtual Environment – Next Generation).**[67] provides a more enterprise-focused alternative, capable of emulating large-scale infrastructures with routers, switches, firewalls, and vulnerable endpoints. Its ability to integrate security appliances makes it particularly valuable for simulating complex attack paths and testing layered defences.
- **Cisco Packet Tracer.**[68] is an educational tool used primarily for basic network configuration and design. While less feature-rich for PenTesting compared to GNS3 or EVE-NG, it is useful for teaching foundational networking concepts and simulating simple attack scenarios.
- **CORE (Common Open Research Emulator).**[69] is another lightweight network emulator frequently used in research and academic contexts. It supports Linux-based nodes and custom topologies, making it a flexible option for small-scale PenTesting experiments.

These network emulation environments provide the backbone for safe and scalable PenTesting labs. They enable security practitioners to build realistic infrastructures that combine attacker platforms, defensive systems, and vulnerable targets, serving as critical components for both manual testing and AI-driven PenTesting research.

[65] https://crp.kypo.muni.cz/
[66] https://www.gns3.com/
[67] https://www.eve-ng.net/
[68] https://www.netacad.com/courses/packet-tracer
[69] https://www.nrl.navy.mil/itd/ncs/products/core

2.3.10 Pre-LLM Approaches to Automated PenTesting

Early work on automated PenTesting, predating the use of LLMs, primarily explored formal decision-making and symbolic planning techniques to reason about attack sequences under uncertainty. A prominent line of research investigated the use of Partially Observable Markov Decision Processes (POMDPs) to model PenTesting as a sequential decision problem with incomplete knowledge of target networks [128, 129]. These approaches demonstrated that principled reasoning about uncertainty could, in theory, guide automated attack selection more effectively than purely heuristic methods. However, their reliance on explicit state-space modelling led to significant scalability challenges, limiting practical applicability to small or highly constrained environments.

Early applications of machine learning (ML) in PenTesting and offensive security were limited and mainly focused on related areas such as intrusion detection rather than autonomous attack execution. Studies such as Sommer and Paxson's analysis of ML for network intrusion detection highlighted fundamental challenges—including concept drift, adversarial behaviour, and class imbalance—that limited the reliability of purely data-driven approaches, reinforcing the need for human expertise in offensive security workflows [142].

Subsequent work shifted towards classical AI planning frameworks to identify multi-step exploitation chains. Pasquale et al. proposed ChainReactor, which employs Planning Domain Definition Language (PDDL) models and lifted planning solvers to discover privilege-escalation paths in containerised systems [38]. ChainReactor operates by exhaustively enumerating system configuration facts, translating them into PDDL representations, and applying manually specified planning rules to infer viable exploitation sequences. While the system successfully identified specific vulnerability classes—such as misconfigured cron jobs and systemd unit files with incorrect permissions—the exploitation itself required manual execution. As a result, these early automated approaches functioned primarily as decision-support or analysis tools for human operators rather than fully autonomous PenTesting systems, highlighting both the potential and the limitations of pre-LLM automation techniques.

Finally, it is worth noting that prior work highlights two fundamental obstacles to full automation [160]: the enormous search space of possible entry points and the strongly target-dependent nature of exploits.

2.3.11 Internet Resources for PenTesting

In addition to formal methodologies, a wide range of Internet resources support PenTesters at every stage of an engagement. Public vulnerability databases such as the National Vulnerability Database (NVD)[70] and Exploit-DB[71] catalogue known CVEs and proof-of-concept (PoC) exploits, while collaborative

[70] `https://nvd.nist.gov/`
[71] `https://www.exploit-db.com/`

platforms like GitHub[72] host scripts, frameworks and updated attack modules. Community-driven knowledge bases, including OWASP[73] and the MITRE ATT&CK Navigator[74], provide structured guidance for web and enterprise environments. Repositories such as GTFOBins[75] and LOLBAS[76] enumerate living-off-the-land techniques for PrivEsc and lateral movement, while HackTricks[77] serves as a constantly updated, practical playbook for offensive security tactics. Online laboratories, including HackTheBox[78], TryHackMe[79] and VulnHub[80], offer realistic training environments, enabling testers to refine their skills against diverse scenarios. Together, these Internet-based resources complement formal standards and commercial tools, forming an indispensable ecosystem for modern PenTesting.

2.3.12 Benchmark Platforms and Experimental Testbeds

The evaluation of ethical hacking techniques—particularly those involving automation or AI—requires carefully selected experimental environments that balance realism, reproducibility, and safety. In practice, researchers and practitioners rely on a diverse ecosystem of platforms ranging from educational CTF challenges and guided training laboratories to highly realistic enterprise network testbeds. Each category serves a distinct purpose: CTF platforms typically provide structured, well-defined tasks suitable for skill development and controlled benchmarking, while enterprise testbeds emulate complex organisational infrastructures and enable the study of post-breach activities such as lateral movement, PrivEsc, and persistence.

Widely used online platforms such as Hack The Box and TryHackMe offer interactive environments with vulnerable machines that approximate real-world scenarios while remaining accessible to learners and researchers. Educational frameworks such as picoCTF and OverTheWire focus on foundational concepts through progressively structured challenges. In contrast, enterprise-grade laboratories—including Active Directory–centric environments such as GOAD and DetectionLab—simulate multi-host corporate networks and are therefore particularly suitable for evaluating advanced offensive techniques and autonomous attack systems.

Table 6 summarises major CTF platforms, hacking practice environments, and enterprise testbeds frequently referenced in cybersecurity training and research.

[72] https://github.com/
[73] https://owasp.org/
[74] https://attack.mitre.org/
[75] https://gtfobins.github.io/
[76] https://lolbas-project.github.io/
[77] https://book.hacktricks.xyz/
[78] https://www.hackthebox.com/
[79] https://tryhackme.com/
[80] https://www.vulnhub.com/

3 Large Language Models

3.1 Overview

LLMs [182] are deep neural networks trained using self-supervised learning on massive text corpora to perform diverse natural language tasks such as generation, reasoning, and summarisation [165]. Most modern LLMs are based on the Transformer architecture [155], which introduced the self-attention mechanism and enabled efficient parallelisation for large-scale sequence modelling.

Model capability correlates strongly with parameter scale. Early models such as GPT-1 (117M parameters) have given way to systems with hundreds of billions or even trillions of parameters. Representative examples include models from the GPT family, Google DeepMind's Gemini models [51], Anthropic's Claude series, Meta's LLaMA family, DeepSeek models, and Microsoft Copilot systems.These systems are widely recognised as valid, cutting-edge LLMs used in both research and production environments.

LLMs function as *foundation models*: pre-trained systems that can be adapted to specific domains via fine-tuning or prompt-based conditioning without the prohibitive cost of training from scratch. The training of GPT-4 reportedly exceeded $100 million in computational resources [29]. This economic reality underscores the growing importance of efficient adaptation methods over full retraining. Open-source frameworks such as `llama.cpp` have further accelerated this by allowing smaller LLMs (up to ∼13B parameters) to run locally without API costs or server-side restrictions, supporting transparent and offline experimentation [53, 152].

LLMs have seen rapid advancements, now capable of executing function calls, reading external documents, and recursively prompting themselves. Together, these capabilities enable LLMs to operate autonomously as agents [169].

Wei et al. [162] observed that certain capabilities emerge only when language models reach a sufficient scale, and cannot be reliably inferred from the performance trends of smaller models. Later research shows that this however needs more investigation [131].

Wang et al. [158] proposed MINT, a benchmark for assessing LLMs' ability to solve tasks through multi-turn interactions that involve both tool use and natural language feedback, revealing that stronger single-turn performance does not necessarily translate into better multi-turn capabilities.

Zheng et al. [183] found that powerful LLMs can reliably act as judges in open-ended evaluations, matching human preferences over 80% of the time — comparable to the agreement level between humans themselves.

3.2 Early ML in Offensive Security

Early applications of ML in PenTesting and offensive security were limited in scope and were primarily explored in adjacent areas such as intrusion detection rather than autonomous attack execution. A seminal example is the work of Sommer and Paxson [142], who critically examined the applicability of ML to

network intrusion detection and highlighted fundamental challenges when moving beyond tightly controlled, "closed-world" assumptions. They argued that real-world network environments exhibit high variability, concept drift, adversarial behaviour, and severe class imbalance, all of which undermine the reliability of purely data-driven models trained on static datasets. These observations significantly influenced early ML-based security research, steering it towards narrow, well-defined tasks such as anomaly detection or traffic classification, and away from end-to-end automation of PenTesting workflows. Consequently, early ML approaches in security complemented human analysts rather than replacing them, reinforcing the view that offensive security required expert reasoning and contextual judgement that traditional ML techniques could not adequately capture.

3.3 LLMs in Cybersecurity

As LLM agents grow more capable, research at the intersection of these models and cybersecurity has expanded rapidly. Studies range from political science analyses predicting whether LLMs will ultimately favour offensive or defensive applications [58, 90, 119] to demonstrations of their use in generating malicious software [114]. LLMs have also been explored in enabling scalable spear-phishing campaigns, with implications for both attack and defence [68, 127].

Kang et al. [79] demonstrated that instruction-following LLMs can be exploited to perform standard security attacks and generate targeted malicious content at a fraction of the human cost, highlighting significant dual-use risks and the need for new mitigation strategies.

3.4 LLM Security

A parallel line of research examines the security of LLMs themselves, focusing on circumventing built-in safeguards designed to prevent the generation of harmful content. This includes a variety of jailbreaking techniques [55, 79, 124, 173, 178, 188].

Collectively, these studies demonstrate that no existing defence mechanism can fully prevent LLMs from producing prohibited content. In our own experiments, we observed that public OpenAI APIs did not block autonomous hacking attempts at the time of writing; should such defences be implemented in the future, existing jailbreaking methods could likely be adapted to bypass them, making this body of work complementary to ours.

3.5 Example LLMs

A variety of LLMs have emerged in recent years, spanning proprietary and open-source ecosystems. Below, we categorise 16 representative models by development family, highlighting their design objectives, technical characteristics, and application domains.

OpenAI Models:

1. **GPT-5 family** [110]. At the time of writing, the GPT-5 series represents the latest generation of OpenAI foundation models and the current frontier of the GPT architecture. Introduced in 2025, GPT-5 integrates improved reasoning, multimodal understanding, and tool-use capabilities compared with earlier GPT systems, and is available in several variants (e.g., GPT-5, GPT-5.1, GPT-5.2, GPT-Auto) designed for different performance and latency requirements.
2. **GPT-4 family** [111]. The GPT-4 generation marked a major step forward in reliability and multimodal capabilities, including models such as GPT-4o that support text, image, and audio inputs. Although widely deployed in earlier applications and research systems, GPT-4 models have largely been superseded by newer GPT-5-series models.
3. **GPT-3.5**. Earlier GPT-3.5 [106] models played an important role in the widespread adoption of LLM-based applications and served as early research and deployment baselines prior to the introduction of GPT-4 and later model generations.

Meta (LLaMA) Models:

4. **LLaMA-2 Chat (7B, 13B, 70B)** [153]. The LLaMA-2 family includes multiple parameter scales and is fine-tuned for conversational use. The 70B model approaches GPT-3.5-level performance, while the 7B and 13B versions are widely adopted for local inference and cost-sensitive applications.
5. **LLaMA-3.1 (70B, 8B)** [153]. The newer LLaMA-3.1 models improve reasoning, instruction alignment, and multilingual support. These models serve as the foundation for newer open-source agentic systems and outperform their LLaMA-2 predecessors on many benchmarks.

Mistral-Based Models:

6. **Mistral-7B Instruct v0.2** [96]. A compact and efficient open-weight model designed for fine-tuning and instruction-following tasks. Its low resource requirements make it popular for embedded and local deployments.
7. **Mixtral-8x7B Instruct** [76]. A mixture-of-experts (MoE) model built from Mistral-7B experts. At inference time, only two of eight experts are active, offering a balance between computational efficiency and performance across diverse NLP tasks.
8. **OpenHermes-2.5-Mistral-7B** [150]. An instruction-tuned variant of Mistral-7B, OpenHermes-2.5 enhances conversational alignment and ranks highly in open-source benchmark leaderboards such as LMSYS Chat Arena.

Yi-Based Models:

9. **Nous-Hermes-2 Yi (34B)** [104]. Built on the Yi-34B base from 01.AI and fine-tuned by Nous Research, this model delivers strong multi-language and code reasoning capabilities, making it a popular choice in high-end open-access deployments.

DeepSeek Models:

10. **DeepSeek-V2 / DeepSeek-Coder-V3** [39]. DeepSeek's latest models include both general-purpose and code-specialised variants, trained on trillions of tokens with high-quality alignment tuning. They are competitive in long-context reasoning and programming benchmarks.

Other Open-Source Models:

11. **OpenChat 3.5** [156]. Designed to mimic ChatGPT's instruction-following behaviour using Direct Preference Optimisation (DPO), OpenChat 3.5 ranks among the top open models in terms of human preference and dialogue coherence.
12. **Falcon 180B** [16]. Developed by the UAE's TII, Falcon 180B is one of the largest openly released models and achieves strong zero-shot performance, particularly in text generation and summarisation tasks.
13. **Command R+** [33]. Cohere's flagship model for Retrieval-Augmented Generation (RAG), Command R+ is optimised for document synthesis, grounded QA, and instruction following in enterprise use cases.

Proprietary Multimodal Models:

14. **Claude 3 / Claude 4** [22]. Developed by Anthropic, the Claude series focuses on constitutional alignment and safety. Claude 4 demonstrates state-of-the-art performance in legal reasoning, multi-step planning, and document analysis.
15. **Gemini 2.5 Pro** [51]. Google DeepMind's Gemini integrates deep retrieval, long-context reasoning, and multimodal support. Version 2.5 Pro adds advanced tool use capabilities and internal memory, making it suitable for autonomous agents.

A recent study indicates that ChatGPT and Google Gemini (formerly Bard) possess comparable ethical hacking capabilities, with Bard slightly outperforming in accuracy, while ChatGPT demonstrates greater comprehensiveness, clarity, and conciseness [125].

Grok Models:

16. **Grok-1 and Grok-1.5** [168]. Developed by xAI, the company founded by Elon Musk, the Grok models are designed to integrate closely with X (formerly Twitter) and prioritise fast reasoning and current event awareness. Grok-1.5 introduced improvements in math, reasoning, and code generation, aiming to rival leading closed models in lightweight settings. While not yet as widely benchmarked in academia, Grok models represent an emerging commercial family optimised for real-time dialogue and autonomous interaction.

These 16 models reflect the diversity of the LLM landscape, spanning a wide range of parameter counts, architectural innovations, and deployment contexts. Proprietary systems dominate in multimodal capabilities and general-purpose performance, while open-source families such as LLaMA, Mistral, Yi, and DeepSeek continue to close the gap through instruction tuning and architectural efficiency.

3.6 Emergence of Locally Executable Open-Source LLMs

Recent advances in open-source LLMs have significantly expanded the options for running high-performance systems locally without relying on proprietary APIs. Meta's LLaMA [152] family, including efficient adaptations such as LLaMA Adapter [181], has demonstrated that large models can be fine-tuned with minimal resources while retaining strong performance on downstream tasks. Stability AI's StableLM suite [143] emphasises lightweight architectures optimised for consumer hardware, enabling experimentation without cloud dependencies. Databricks' Dolly 2 [34] further illustrated the potential of instruction-tuned models trained on transparent, open datasets to produce commercially viable conversational agents. Complementing these efforts, Koala [52], developed at UC Berkeley, focused on alignment through carefully curated academic and dialogue data to produce a research-friendly alternative to closed systems.

A key advantage of these locally executable models is that they incur no ongoing cloud costs and mitigate privacy concerns by avoiding the transmission of sensitive data to external servers. These properties make locally run LLMs particularly attractive for highly regulated domains such as finance, healthcare, and government, where cost control and strict data governance are critical requirements.

Together, these projects underscore a shift towards democratised LLM development, prioritising local execution, open weights, and reproducible training pipelines to support both academic research and applied AI development.

3.7 LLM Agents

LLM agents [95,169] are systems that integrate a LLM within a reasoning–acting loop [175], enabling the model to operate autonomously or semi-autonomously across multiple steps to achieve a specified objective. In this paradigm, the LLM is no longer limited to single-turn prompt–response interactions; instead, it is embedded within an architecture capable of perceiving inputs, generating plans, invoking external tools [115], and iteratively refining its actions based on intermediate feedback [139, 159, 169].

Although no universally agreed definition of LLM agents exists, they are generally characterised as systems that employ a LLM to reason about a problem, formulate a plan, and execute that plan through interactions with external tools and environments [132, 154, 169]. A typical LLM agent framework consists of four main components:

1. **perception**, in which the agent receives information from the environment or the user;
2. **planning**, where the LLM generates a sequence of steps or subgoals, often leveraging chain-of-thought (CoT) reasoning [163];
3. **action**, involving the execution of commands or tool calls (e.g., APIs, search engines, or code execution environments); and
4. **reflection**, where the agent assesses the results of its actions and determines subsequent steps [175].

This loop allows LLM agents to handle complex, multi-turn tasks that require integrating information from prior steps and adapting strategies dynamically.

One of the most widely adopted frameworks for building such agents is LangChain [32], an open-source software toolkit for developing applications powered by LLMs, providing tools for prompt management, chaining, memory, and integration with external data sources and APIs.

Recent research has explored the potential of LLM agents across diverse domains, including software engineering [77, 172], scientific research [27] & discovery [28], and cybersecurity [6, 48]. Benchmarks such as MINT [158] and MT-Bench [183] have been developed to evaluate their multi-turn reasoning and tool-use capabilities.

Despite their promise, LLM agents present significant challenges. They are prone to error accumulation over multiple turns, may exhibit hallucinated actions, and can be exploited for malicious purposes through standard attack vectors [55, 79]. Furthermore, their performance can be highly sensitive to prompt design, environmental constraints, and the quality of integrated tools. These limitations, combined with the computational cost of sustained multi-turn operation, underscore the need for robust evaluation frameworks, safety mechanisms, and cost–benefit analyses before large-scale deployment.

3.8 Efficient Alternatives to Full LLM Training

Training LLMs from scratch incurs substantial computational and financial costs, prompting a surge of alternative strategies aimed at achieving competitive performance without full-scale retraining. One such paradigm is *In-Context Learning* (ICL), where background knowledge is embedded directly into the prompt, effectively substituting internalised model parameters with externally supplied information [42]. A related technique, CoT [163] prompting, augments the context with step-by-step reasoning traces to guide the model towards structured outputs [82]. Both methods shift the burden of knowledge representation into the prompt itself, making the available context window a critical and scarce resource—even as successive model generations offer increasingly large context sizes.

Complex, real-world tasks often require decomposition into a sequence of subtasks, motivating a line of research on hierarchical or iterative reasoning with LLMs. Lightweight frameworks such as *BabyAGI* [102] explore minimal

autonomous task chaining, while more structured approaches such as *Tree-of-Thoughts* [174] and *Task-List*-based planning [40] explicitly manage branching reasoning paths and execution order. Wang et al. [157] describe this family of techniques under the umbrella of *plan-and-solve*, emphasising the integration of task planning with adaptive problem solving in order to extend LLM capabilities beyond single-turn inference.

4 Emerging Tactics for LLM-driven PenTesting Systems

This section outlines the core prompting strategies and optional enhancements underpinning `PenTest2.0`, divided into five components: the base prompt, CoT reasoning, Human Hint injection, RAG, and PTT-based task tracking.

4.1 PenTest Task Tree (PTT) Tracking

The PTT, introduced in `PenTestGPT` [40], is a lightweight task-tracking mechanism designed to maintain a structured representation of a PenTesting workflow across multiple interaction turns. In LLM-assisted PenTesting systems, where the model repeatedly analyses system states and proposes follow-up actions, maintaining such structured task memory is essential for preserving context, avoiding redundant suggestions, and tracking progress during complex attack sequences.

Conceptually, the PTT organises the PenTesting process into a hierarchy of objectives and subtasks, each associated with a status indicator (e.g., pending, completed, or skipped). As the interaction progresses, the system updates the task tree to reflect newly discovered attack opportunities, completed steps, or unsuccessful attempts. By explicitly representing the evolving attack plan, the PTT enables the LLM to reason more coherently about prior actions and future steps, thereby improving the consistency and efficiency of multi-turn PenTesting workflows. Such structured PTT summaries allow the model to retain awareness of the current testing strategy while adapting to new information obtained during command execution.

Recent LLM-aided PenTesting systems extend this idea to support structured reasoning and task tracking across multi-turn engagements, including `PenTest2.0` (Section 5.2), `Cochise` (Section 5.14), and other LLM-driven PenTesting systems incorporating hierarchical task planning (see Section 7.4).

4.2 Prompt Engineering

A critical dimension in leveraging LLMs is *prompt engineering*, the practice of crafting input instructions that guide LLM behaviour towards desired outputs [145]. In conversational systems such as ChatGPT and Copilot, interactions follow a prompt–response loop in which even small changes in wording, context, or structure can significantly affect output quality, reasoning performance, and safety. Consequently, a variety of prompting strategies have been developed to

improve model reliability, including few-shot prompting, CoT reasoning, and self-consistency techniques [41].

Modern LLM interfaces also support higher-level conditioning mechanisms. For example, GPT-4 and GPT-4o introduced *system messages*, which allow developers to define the tone, role, and behavioural constraints of the model prior to user interaction. Such prompt conditioning can act as a lightweight form of alignment, enabling practitioners to shape model behaviour without modifying model weights. Research has shown that in specialised domains such as cybersecurity, prompt design can significantly influence both exploit-chain success and the reliability of defensive analyses [64]. This shift illustrates an important paradigm change: instead of relying solely on model retraining, practitioners increasingly achieve task specialisation through careful prompt design, enabling rapid experimentation and deployment.

Within LLM-driven security systems, prompt engineering sometimes includes the construction of a *base prompt*, which defines the operating context, rules of engagement, and expected input/output structure for the model. For example, in `PenTest2.0` (see Section 5.2), the base prompt establishes the simulated environment as a real-time penetration test and imposes strict behavioural constraints, including the use of non-interactive commands, adherence to structured outputs, and clear reasoning. An excerpt from the base prompt used in [11] is shown below.

Excerpt from the Base Prompt in [11]

```
You are assisting with Linux Privilege Escalation. You
have a low-level privileged account (USERNAME), and your
ultimate goal is to elevate your privilege access to
root level. We will interact with you turn by turn: at
each step, you'll receive a summary of the target system
context and you must suggest the next best command. If
root is not yet achieved, we will send you the execution
output. Your goal is to gain root access within a maximum
of MAX_TURNS turns.
...
Rules You MUST Follow:
1. NEVER repeat a command that already succeeded
*within the current run*. You MAY suggest it again in
future sessions if the target system appears similar or
identical.
2. You MUST retry a corrected version of a previous
command that failed due to syntax errors or
misconfiguration. Do not suggest the exact same faulty
command again.
3. NEVER suggest dangerous or destructive commands,
including:
- rm -rf *, rm -rf /, dd if=/dev/zero
- zip -r backup.zip /, zip -r backup.zip /etc
```

```
- Any command involving heavy backups or full system
traversal
4. NEVER break the JSON structure, even if unsure.

...
```

The base prompt also specifies a **mandatory** output structure to ensure that responses remain both machine-readable and operationally safe. In this example, the LLM must return a single compact JSON object containing the suggested command, contextual reasoning, and an updated summary of the system state, as illustrated below.

```
{
 "command_non_interactive": "string, safe for automated execution (no $, #, `)",
 "command_interactive": "string, interactive version if applicable, else empty",
 "system_summary": "string, max 10 very short bullet points",
 "command_history": "string (max 15 lines, summarised cleanly)",
 "rationale": "string, 1-2 sentences, explaining why this command was chosen"
 {% if rag_enabled %}
 ,
 "rag_search_query": "string (max 15 words)"
 {% endif %}
 {% if ptt_enabled %}
 ,
 ...
 {% endif %}
}
```

This structured prompting approach ensures that LLM outputs remain syntactically valid and semantically appropriate for automated execution, thereby improving reproducibility, reliability, and operational safety in LLM-assisted penetration testing workflows.

4.3 Chain-of-Thought (CoT)

CoT prompting is a widely studied technique for enhancing the reasoning capabilities of LLMs by encouraging them to generate intermediate reasoning steps before producing a final output [157, 163]. Instead of immediately generating an answer, CoT decomposes the task into a sequence of reasoning steps, thereby enabling more deliberate and interpretable behaviour. This capability is particularly valuable in complex domains such as PenTesting, where outputs must be interpreted in context and reasoning often evolves across multiple interaction turns.

In practice, CoT prompting is typically implemented by adding an instruction that explicitly encourages step-by-step reasoning. For example, in LLM-assisted security workflows the prompt may instruct the model to first analyse the available system context, then evaluate prior commands and their outputs, and finally determine the most appropriate next action. Such instructions encourage the model to reflect on prior states and reasoning traces before producing the next command or recommendation, improving both transparency and robustness during multi-step interactions.

Several variants of CoT prompting have been proposed to elicit structured reasoning from LLMs.

- **Zero-shot CoT:** In this approach, the model is encouraged to produce intermediate reasoning steps using a simple instruction such as "`think step by step`". No demonstrations or examples are provided; instead, the model relies on knowledge acquired during pretraining to generate a reasoning trace leading to the final answer.
- **Few-shot CoT:** Few-shot CoT prompting extends this idea by including a small number of exemplar reasoning traces within the prompt. These demonstrations illustrate how a task can be decomposed into intermediate reasoning steps and how conclusions should be derived from them. By observing such examples, the model can better emulate structured reasoning when solving new problems.
- **Fine-tuned CoT:** In contrast to prompt-based approaches, fine-tuned CoT models are explicitly trained on datasets containing annotated intermediate reasoning steps [161]. During training, the model learns to generate reasoning traces alongside final answers, enabling more consistent multi-step reasoning behaviour at inference time.

In the context of LLM-assisted PenTesting, CoT prompting can help models analyse system states, reason about possible attack paths, and determine appropriate follow-up actions across multiple steps. Several recent systems for LLM-driven security automation adopt such multi-step reasoning strategies, often combining CoT-style prompting with reasoning–acting loops or agent-based architectures to iteratively analyse targets, generate commands, and interpret execution feedback [11, 65, 175].

4.4 Retrieval-Augmented Generation (RAG)

RAG [87] enhances the reasoning capabilities of LLMs by grounding their outputs in external knowledge sources. Instead of relying solely on knowledge internalised during pretraining, RAG augments the generation process by retrieving relevant information from external corpora and incorporating it into the model's prompt context. This retrieved information can include technical documentation, exploit references, configuration examples, or other domain-specific artefacts that help the model reason more effectively about the task at hand.

In cybersecurity and PenTesting applications, RAG can be particularly valuable because the knowledge required to analyse vulnerabilities or exploit misconfigurations often depends on up-to-date technical resources. By providing the LLM with relevant external guidance, RAG helps reduce hallucinations, improve factual grounding, and align model outputs with real-world system behaviour.

RAG-based architectures typically rely on a retrieval pipeline in which relevant documents or snippets are first identified using keyword search or vector similarity techniques (e.g., embedding-based retrieval using vector databases

such as FAISS[81]). The retrieved content is then incorporated into the prompt, allowing the LLM to reason in context using both its pretrained knowledge and the retrieved material.

For example, a prompt may include retrieved information such as:

> "Retrieved Insight: GTFOBins suggests that `sudo tar` can spawn a shell using the `--checkpoint-action=exec=` option."

By injecting such contextual knowledge into the reasoning process, RAG enables LLM-driven systems to operate with greater situational awareness and technical grounding. As a result, RAG has become a common architectural component in modern LLM-assisted security analysis and PenTesting frameworks, where accurate interpretation of system states and exploitation techniques is essential.

4.5 Large Reasoning Models (LRM)

Reasoning LLMs, often referred to as LRMs, represent a recent class of models that are explicitly trained to internalise multi-step reasoning processes rather than relying on external prompt-based elicitation of CoT behaviour. Unlike earlier approaches that depend on prompting techniques such as explicit step-by-step instructions, LRMs incorporate structured reasoning during training, enabling them to perform extended deliberation over complex and ambiguous tasks [107]. OpenAI's initial reasoning model (o1-preview) was announced in late 2024 and subsequently made available through public APIs, alongside comparable models such as Alibaba's Qwen3 and DeepSeek's R1 [15, 56, 107].

A key implication of this training paradigm is that many established prompt-engineering strategies developed for non-reasoning LLMs no longer yield consistent benefits. Empirical evidence suggests that manually injecting CoT instructions into reasoning models can degrade instruction-following performance, rather than improve it [88]. Practitioner-oriented guidance similarly reports that few-shot prompting should be used sparingly with LRMs, as excessive demonstrations may interfere with internally learned reasoning mechanisms [108]. This marks a departure from earlier LLM usage patterns and necessitates revised design principles for agent-based systems.

Recent evaluations further indicate that the advantages of LRMs are task-dependent rather than universal. Complementary findings by Shojaee et al. [137] show that on simpler tasks, non-reasoning models may outperform LRMs due to reduced over-deliberation, whereas on moderately complex tasks, LRMs can achieve superior results through more systematic exploration; on highly complex tasks, both model classes may fail. These results caution against assuming monotonic improvements from reasoning-centric training.

[81] FAISS (Facebook AI Similarity Search) is an open-source library for efficient similarity search and clustering of dense vectors, widely used to enable fast retrieval of semantically relevant content in AI systems and RAG setups.

Within the context of PenTesting and offensive security, reasoning models are particularly relevant to tasks involving strategic planning, hypothesis refinement, and action selection under uncertainty. Happe and Cito argue that many PenTesting activities rely on transferable patterns learned through prior experience (e.g., CTF exercises), while also requiring structured decision-making when navigating realistic enterprise environments such as Microsoft Active Directory networks [62, 65].

Consequently, an emerging architectural pattern is to employ reasoning models primarily for high-level planning and decision-making, while delegating concrete execution tasks to lower-latency, non-reasoning LLMs. This division of labour aligns with recent industrial guidance that distinguishes between models optimised for deliberation and those optimised for efficient task execution [108]. From a system-design perspective, this hybrid approach offers a pragmatic balance between accuracy, cost, and responsiveness, while mitigating some of the limitations observed when reasoning models are applied indiscriminately.

4.6 Human Hint Injection

In many PenTesting scenarios, human operators possess domain expertise, contextual knowledge, or prior reconnaissance results that may not be fully captured within the LLM's prompt context. *Human hint injection* refers to the structured incorporation of such expert guidance into the prompt to influence the model's reasoning and decision-making process. This approach aligns with broader Human-in-the-Loop (HITL) design principles, where automated systems remain guided by human expertise to improve reliability, safety, and task performance [20, 89].

Rather than allowing the model to operate entirely autonomously, hints provide targeted cues that can steer the reasoning process toward more relevant attack paths, commands, or diagnostic actions. These hints may reflect operator intuition, prior observations, or constraints discovered during earlier testing phases.

For example, a prompt may incorporate guidance such as:

> "Human Hint: Use the `id` command instead of `/bin/sh` to verify root privileges in an automated workflow."

By combining automated reasoning with selective human guidance, this technique can improve exploration efficiency, reduce repeated mistakes, and support more controlled multi-step PenTesting workflows.

4.7 Human-in-the-Loop (HITL) Architectures

HITL architectures incorporate human oversight into the operational workflow of AI systems, enabling collaboration between automated reasoning and expert supervision. Rather than delegating all decisions to the LLM, HITL designs embed control points where human operators can monitor system behaviour,

validate outputs, or intervene when necessary. Such architectures are widely used in safety-critical domains to maintain accountability, interpretability, and operational control in AI-assisted decision-making processes [20, 89].

Within LLM-assisted cybersecurity and PenTesting systems, HITL architectures typically introduce supervisory stages around automated reasoning loops. Human operators may review intermediate results, approve potentially sensitive actions, or redirect the exploration strategy when automated reasoning becomes unreliable. This layered design allows AI components to accelerate analysis and command generation while ensuring that critical decisions remain under expert control, thereby improving robustness and operational safety in complex multi-step security assessments.

4.8 Pre-trained autonomous-agent frameworks

Early general-purpose agents, including those listed here, illustrate how LLMs can orchestrate complex tasks using self-generated prompts and multi-agent collaboration.

AutoGPT. AutoGPT [139] is an open-source prototype that seeds an LLM with a high-level user goal and then lets the model iteratively draft and refine its own prompts. By augmenting the LLM with web searches and optional human feedback [112], the system decomposes an initial goal into a list of subtasks, reducing manual prompt engineering and partially mitigating hallucinations.

BabyAGI. BabyAGI [102] takes this further by organising work into a queue of subtasks: a task-creation module uses an LLM to generate new subtasks, a context module retrieves relevant information from memory, and task-execution and prioritisation modules dispatch and reorder the queue. Simplified versions of BabyAGI demonstrate that such pipelines can be implemented in only a few dozen lines of Python code.

HuggingGPT. Finally, Jarvis (also known as HuggingGPT [136]) composes multiple models—including ChatGPT and domain-specific HuggingFace models—into a multimodal, multi-agent system that delegates tasks to the most capable component.

Collectively, these prototypes show that coupling LLMs with retrieval mechanisms, feedback loops and specialised sub-agents can make them more autonomous and reduce hallucinations, providing a conceptual foundation for later PenTesting agents.

5 LLM-aided PenTesting Systems

This section surveys 27 representative LLM-aided PenTesting systems, including peer-reviewed papers (see Table 7) and recent preprints (see Table 8), capturing the current state of AI-driven offensive security research.

5.1 PenTest++

5.1.1 Introduction

PenTest++ [6,7] is an AI-augmented, command-line PenTesting system designed to automate core ethical-hacking tasks while maintaining strong human oversight. The framework addresses longstanding limitations of traditional PenTesting, including dependence on expert operators, time-intensive workflows, and the cognitive burden of recalling complex commands across diverse tools and environments.

Unlike fully autonomous attack agents, PenTest++ adopts a mixed-initiative paradigm in which automation and GenAI assist the user but do not replace human decision-making. The system integrates a LLM to interpret tool outputs, recommend attack strategies, and generate documentation, thereby streamlining reconnaissance, scanning, exploitation, and reporting activities.

The work is presented as a PoC demonstrating how AI can enhance efficiency, accessibility, and scalability of ethical hacking while emphasising the necessity of human validation to mitigate risks such as hallucinations and misuse.

5.1.2 Architecture and Workflow

PenTest++ follows a modular pipeline aligned with conventional PenTesting phases. Each stage combines automated tool execution with GenAI-assisted analysis and user approval.

- **Reconnaissance:** The system automatically discovers live hosts by determining the attacker's subnet and performing network scans (e.g., using `nmap -sn`). Users select a target from detected hosts.
- **Scanning and Enumeration:** Comprehensive scans identify open ports, services, and potential vulnerabilities. Outputs are parsed and presented in structured tables, while the LLM correlates findings with known weaknesses and suggests further enumeration steps.
- **Exploitation:** For detected services (e.g., FTP, HTTP, SSH, NFS), the system applies tailored attack strategies. GenAI provides guidance on payload generation, credential analysis, and attack sequencing, while users approve actions and can supply additional inputs.
- **Documentation:** PenTest++ automatically generates structured PenTesting reports using GenAI, including methodology, findings, risk assessments, and recommendations, output in multiple formats (e.g., text, JSON, PDF).

A key design principle is transparency: executed commands are displayed to the user, ensuring trust and enabling manual intervention at any stage.

5.1.3 Prototype Implementation and Case Studies

The authors present a prototype implemented in Python and evaluated in a controlled virtual laboratory comprising a Kali Linux attacker VM and multiple

Linux target machines. The system integrates external security tools (e.g., `nmap`, `gobuster`, `hashcat`, `hydra`) alongside an online LLM accessed via API.

Two case studies demonstrate end-to-end exploitation:

- **Target VM 1:** Anonymous FTP access exposed credentials that enabled web-application authentication. An insecure file upload mechanism was exploited to deploy a reverse shell, achieving system access.
- **Target VM 2:** Misconfigured NFS shares revealed sensitive files, including a password-protected archive containing an SSH private key. Subsequent web enumeration uncovered credentials and a local file inclusion (LFI) vulnerability, ultimately enabling SSH compromise.

These scenarios illustrate how PenTest++ coordinates multiple attack vectors while preserving user control over decisions.

5.1.4 Strengths

PenTest++ provides several key features.

- **Mixed-Initiative Automation:** Combines automated command execution with human approval, reducing workload while maintaining oversight.
- **GenAI-Assisted Analysis:** LLMs interpret complex tool outputs, extract actionable intelligence, and recommend attack strategies.
- **Modular Architecture:** Supports integration of additional tools and attack techniques across platforms.
- **Automated Reporting:** Produces structured PenTesting reports directly from execution logs.
- **User-Centric Design:** Maintains transparency by displaying commands and allowing manual adjustments.

5.1.5 Limitations

The authors acknowledge several constraints:

- **Partial Automation:** The system does not achieve full autonomy; key decisions require user validation.
- **Limited Scope:** Post-exploitation tasks such as PrivEsc and persistence are not addressed.
- **Controlled Evaluation Environment:** Experiments are conducted in laboratory settings, limiting generalisability.
- **Privacy and Ethical Concerns:** Sending data to external LLM services raises confidentiality and legal issues.
- **LLM Reliability Risks:** Hallucinations and incorrect recommendations remain an ongoing concern, necessitating human oversight.

5.1.6 Summary

PenTest++ demonstrates a pragmatic approach to AI-assisted PenTesting that emphasises human-AI collaboration rather than full autonomy. By integrating automation, external security tools, and LLM-based analysis within a modular workflow, the system reduces operational complexity while preserving transparency and ethical control. The work highlights both the promise and the practical constraints of deploying AI-augmented offensive security tools, suggesting that mixed-initiative designs may offer a realistic pathway toward scalable and trustworthy automated PenTesting.

5.2 PenTest2.0

5.2.1 Introduction

PenTest2.0 [8, 11] is an LLM-driven post-exploitation framework that focuses specifically on automating multi-turn Linux PrivEsc with strong operator governance. It is positioned as a major evolution of the authors' earlier PenTest++ system, which automated reconnaissance, scanning, exploitation, and reporting but did not cover PrivEsc. PenTest2.0 extends that workflow by taking over after an initial low-privilege foothold is obtained, and repeatedly reasoning, proposing commands, executing them, and adapting based on execution feedback.

To improve robustness and traceability in long command-based PrivEsc sessions, PenTest2.0 employs four optional mechanisms: *CoT prompting*, *RAG*, *PTTs*, and *human hints*. The authors emphasise that the system remains under human control: before each LLM call the user approves the full prompt (including estimated token cost), and before each command execution the user explicitly approves the proposed command.

5.2.2 Architecture and Workflow

PenTest2.0 is organised around an iterative *execution–feedback* loop (illustrated in Fig. 4), in which the system captures state, builds a prompt, obtains an LLM command suggestion, executes via SSH, checks for root, and repeats until success or a turn limit.

The workflow comprises the following stages:

1. **Prior Assumption (Foothold):** The system assumes an existing low-privilege shell (post-exploitation setting), and constrains scope to PrivEsc.
2. **System Context Collection (Probing):** Before the first LLM turn, PenTest2.0 runs a predefined probing set (e.g., `id`, `whoami`, `uname -a`, `sudo -l`, environment checks, temporary directory checks). Outputs are condensed into an initial system snapshot to seed LLM reasoning.
3. **Prompt Construction:** Prompts are composed dynamically from reusable blocks (system facts, recent outputs, command history, and optional CoT/RAG/PTT/hints). A strict JSON response schema is enforced so the system can parse outputs reliably.

4. **LLM Suggestion and Dual-Format Commands:** The LLM must return a *non-interactive* command for automated SSH execution, plus an optional interactive variant for manual use. This is intended to prevent hangs from interactive shells during automation.
5. **User Oversight and Safety Controls:** Each turn has two approval checkpoints: (i) approve the prompt and its estimated API cost, and (ii) approve the command prior to SSH execution. In addition, a local blacklist filters unsafe patterns (e.g., destructive or resource-heavy commands) before user review.
6. **Root Detection and Termination:** After execution, the system applies regex-based checks (e.g., detecting `uid=0(root)`) to confirm escalation and terminate early on success.
7. **Logging and Reporting:** PenTest2.0 logs prompts, commands, outputs, token usage, costs, and diagnostics, producing structured session summaries to support reproducibility and analysis.

5.2.3 Optional Enhancements

PenTest2.0 exposes four feature toggles intended to improve success, traceability, or both:

- **CoT Prompting:** A lightweight directive (zero-shot) or concise examples (few-shot) encourage stepwise reasoning before selecting the next command, aiming to reduce repetition and improve decision quality.
- **Human Hints:** Operators can inject short guidance (e.g., recommending a safer verification method) to steer the LLM away from ineffective behaviours with minimal prompt overhead.
- **RAG (Offline + Online):** A hybrid approach combines locally indexed security knowledge (e.g., GTFOBins content) with optional online retrieval, injecting short snippets only when needed to ground recommendations and reduce hallucinations.
- **PTT Tracking:** The PTT records subtasks, statuses, command history, and "commands to avoid", enabling persistent task-level context across turns and improved traceability of the escalation process.

5.2.4 Evaluation

The authors evaluate PenTest2.0 on a vulnerable Linux target VM under seven configurations combining or excluding CoT, hints, RAG, and PTT, with a maximum of ten turns per run. All seven configurations ultimately achieved root in the test environment, but performance differed substantially. In particular, *CoT+Hint* achieved root in a single turn, while several heavier configurations (No-Flags, RAG, PTT) reached the 10-turn cap and required manual confirmation due to limitations in automated root detection.

A key observed failure mode is that some effective PrivEsc techniques spawn an interactive root shell (e.g., `sudo awk 'BEGIN system("/bin/sh")'`)

which the non-interactive SSH wrapper cannot easily observe; this can cause the system to miss "auto-root" even when escalation succeeds. The paper highlights this as a practical limitation and an area for future improvement.

5.2.5 Cost Analysis

PenTest2.0 includes explicit cost tracking per turn (based on OpenAI pricing used by the authors) and shows that *more verbose or feature-heavy prompting does not necessarily improve outcomes.* Early-success configurations (CoT+Hint, Hint, CoT) achieved escalation in 1–2 turns at very low cost, whereas long-running configurations (No-Flags, RAG, PTT) consumed more tokens and cost without improving effectiveness. The authors conclude that lightweight prompting, particularly CoT combined with human hints, offers the best speed–cost trade-off in their setting.

5.2.6 Strengths

PenTest2.0 provides several key features.

- *Multi-turn, execution-grounded PrivEsc:* An iterative loop that adapts strategy using real command outputs rather than single-shot suggestions.
- *Strong governance model:* Dual user approval (prompt+cost approval, then command approval) combined with command blacklisting.
- *Composable prompting:* Dynamic prompt assembly enables cost-efficient minimal prompts with optional escalation to CoT/RAG/PTT/hints.
- *Traceability and reproducibility:* Detailed logging of prompts, commands, outputs, and token/cost metrics supports auditing and scientific evaluation.
- *Engineering pragmatism:* Non-interactive command enforcement and structured JSON outputs improve reliability for automated execution.

5.2.7 Limitations

The paper identifies multiple practical limitations typical of LLM-driven command agents, as follows.

- *Prompt sensitivity and semantic drift:* Long sessions can degrade effectiveness as prompts grow and the model repeats unproductive strategies.
- *Unsafe or costly suggestions:* LLMs may propose destructive or resource-intensive commands, requiring filtering and human oversight.
- *Interactive-shell detection gap:* Some successful PrivEsc methods bypass automated root detection when they spawn interactive shells.
- *Generalisation limits:* Evaluation is performed on controlled Linux targets with known PrivEsc vectors, limiting generalisability.
- *Operational constraints:* Reliance on cloud-hosted LLMs introduces privacy/compliance concerns for real organisational deployments.

5.2.8 Summary

PenTest2.0 advances AI-assisted PenTesting by extending automation into the post-exploitation phase, demonstrating that LLM-guided, multi-turn Linux PrivEsc can be executed in an execution-grounded and auditable manner when paired with strict operator control. Its key insight is that lightweight reasoning support (CoT) and minimal human steering (hints) can substantially improve convergence and cost efficiency, whereas heavier context mechanisms (RAG/PTT) may increase overhead without guaranteeing better outcomes. The system surfaces important open challenges for LLM-driven offensive tooling, including safe command governance, interactive-shell handling, and robustness against drift in longer sessions.

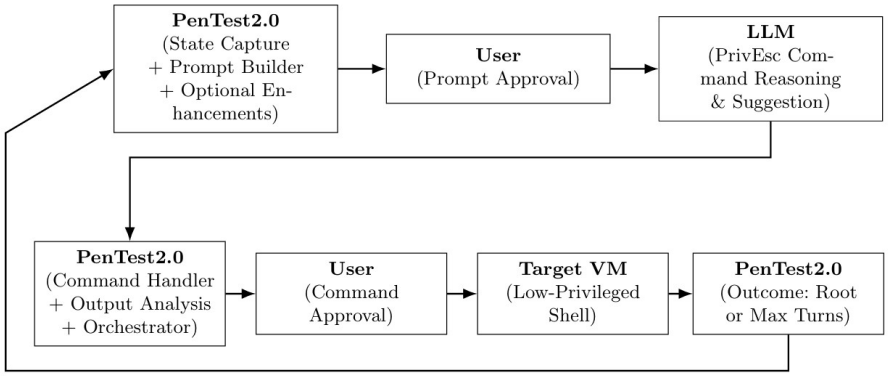


Fig. 4: High-level architecture of PenTest2.0, reproduced from [11] under fair use

5.3 WiFiPenTester

5.3.1 Introduction

WiFiPenTester [10] is a governed GenAI-assisted system for wireless PenTesting that integrates LLMs into the reconnaissance and decision-support phases of IEEE 802.11 security assessment while maintaining strict human oversight. Unlike prior automation tools that rely on static heuristics or operator intuition, the system leverages LLM reasoning to prioritise targets, estimate attack feasibility, and recommend strategies based on structured scan metadata.

The system is motivated by the labour-intensive and error-prone nature of traditional wireless ethical hacking, where practitioners must manually interpret radio-frequency (RF) conditions, select viable targets, and coordinate time-sensitive operations such as handshake capture. WiFiPenTester addresses these

challenges through a governance-first design that emphasises bounded autonomy, auditability, reproducibility, and explicit HITL control. Experimental results indicate that GenAI assistance can improve target selection accuracy and operational efficiency while preserving ethical safeguards and legal compliance.

5.3.2 Architecture and Workflow

WiFiPenTester employs a modular architecture that augments conventional wireless toolchains (e.g., Aircrack-ng utilities) with structured GenAI decision support. The workflow proceeds through a sequence of governed stages, as follows (see Fig. 5).

- *Passive Reconnaissance:* The system performs monitor-mode scanning to enumerate nearby access points (APs), collecting metadata such as BSSID/ESSID, encryption type, channel, received signal strength indicator (RSSI), client activity, and protocol features.
- *Structured Metadata Aggregation:* Observations are normalised into a deterministic internal representation to ensure that LLM reasoning is grounded in factual data rather than free-form logs.
- *Prompt Construction and Budget Gate:* Aggregated data are injected into a constrained prompt that assigns the model an expert role, restricts output to advisory reasoning, and enforces a fixed JSON schema. Token usage and estimated cost are computed, requiring explicit user approval before submission.
- *LLM-Based Target Ranking:* The model produces a structured assessment including ranked candidate networks, feasibility scores, justifications, and recommended strategies.
- *HITL Target Selection:* The operator reviews both raw scan data and model recommendations before selecting a target, ensuring that GenAI advice never directly triggers actions.
- *Controlled Active Operations:* Only after approval does the system perform channel locking, optional deauthentication, handshake capture, and protocol-specific assessment using deterministic tools.
- *Evidence Consolidation and Reporting:* All artefacts—including scan data, prompts, responses, command traces, and outcomes—are archived, and a final GenAI-assisted report is generated from sanitised facts.

A key architectural principle is strict separation between reasoning and execution: the LLM provides analysis only, while all RF operations are performed by conventional tools under human control.

5.3.3 Evaluation

The prototype was implemented in Python on a Kali Linux testbed using commodity hardware and external wireless adapters supporting monitor mode and

packet injection. Experiments across controlled wireless environments demonstrate that GenAI assistance can effectively prioritise targets based on factors such as signal strength, client presence, and authentication configuration.

In tested scenarios, the system successfully captured WPA2 handshakes following LLM-guided target selection and recovered passphrases via offline dictionary attacks. The evaluation also highlights sensitivity to environmental dynamics, including fluctuating signal conditions and client behaviour, which can affect attack feasibility across runs. Overall, the results indicate that LLM-driven decision support improves efficiency while remaining probabilistic and dependent on human validation.

5.3.4 Strengths

WiFiPenTester contributes several design innovations for safe GenAI integration into offensive wireless security workflows:

- *Governance-First Design:* Bounded autonomy with mandatory HITL checkpoints ensures ethical and accountable operation.
- *Structured Reasoning over Metadata:* Decisions are based on normalised observations, reducing hallucination risk and improving reproducibility.
- *Auditability and Evidence Preservation:* Comprehensive logging of prompts, responses, costs, and actions enables experimental validation.
- *Budget-Aware Execution:* Explicit cost estimation prevents uncontrolled API usage and supports resource management.
- *Modular Extensibility:* Separation of wireless interaction, reasoning, and reporting components facilitates adaptation to new protocols and models.

5.3.5 Limitations

The authors identify several constraints:

- *Dependence on Environmental Data Quality:* Incomplete or transient reconnaissance data may lead to suboptimal recommendations.
- *Operational Disruption Risks:* Active wireless attacks can affect third-party devices, necessitating strict governance.
- *Privacy Considerations:* Use of cloud-based LLMs may expose sensitive metadata unless local models are employed.
- *Limited WPA3 Support:* Current evaluation focuses primarily on WPA/WPA2-PSK scenarios.
- *LLM Reliability Issues:* Performance remains sensitive to prompt design and model behaviour.

5.3.6 Summary

WiFiPenTester demonstrates a principled approach to integrating GenAI into wireless PenTesting by confining LLM capabilities to advisory reasoning while

maintaining deterministic execution under human supervision. By combining structured RF reconnaissance with governed decision support, the system illustrates how LLMs can enhance efficiency and consistency in complex cyber-physical environments without relinquishing control. The work highlights both the promise of AI-assisted wireless security assessment and the necessity of rigorous safeguards for trustworthy deployment.

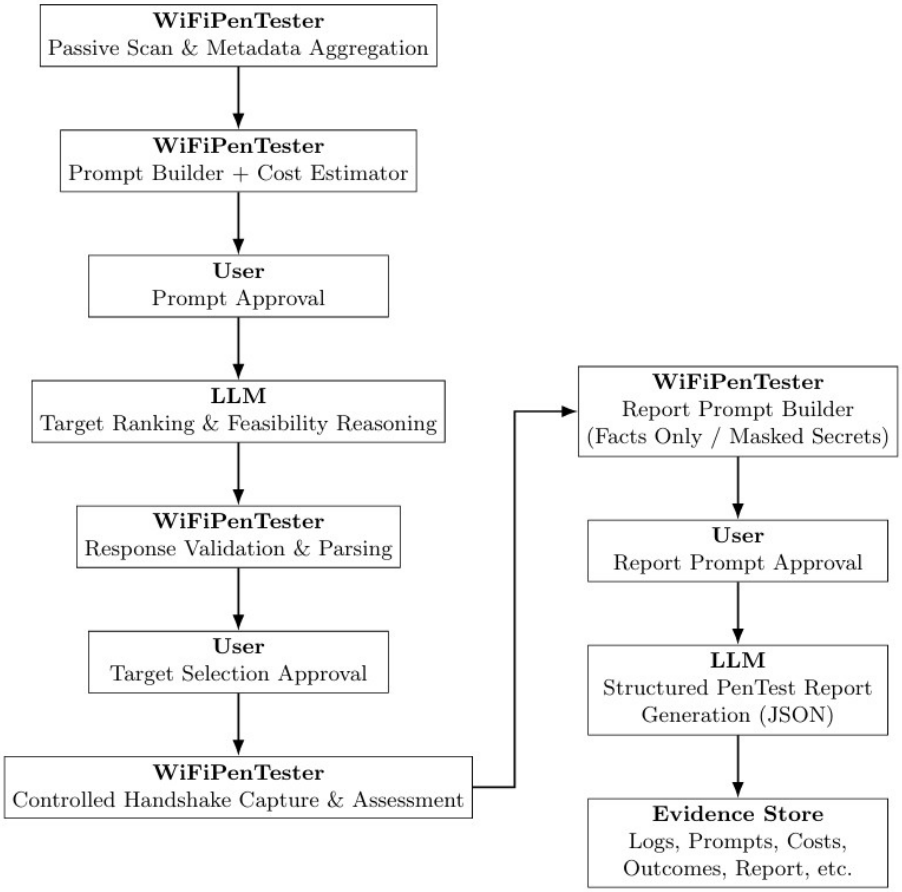


Fig. 5: High-level architecture of WiFiPenTester, reproduced from [10] under fair use

5.4 PentestGPT

5.4.1 Introduction

PentestGPT [40] is an LLM-empowered framework for automated PenTesting that structures model reasoning to support multi-stage offensive workflows. The authors show that although LLMs can perform individual tasks—such as command generation, vulnerability analysis, and tool usage—they struggle to complete full engagements due to context loss, inconsistent planning, and limited situational awareness. To address these shortcomings, PentestGPT introduces a coordinated architecture designed to maintain long-term context and guide attack progression across multiple stages.

The system is evaluated using a benchmark derived from HackTheBox and VulnHub machines, decomposed into 182 subtasks spanning OWASP Top 10 vulnerabilities and several CWE categories. Results indicate that naïve LLM usage performs reasonably on isolated tasks but degrades significantly on complex targets, motivating a structured orchestration approach.

5.4.2 Architecture and Workflow

PentestGPT adopts a modular design inspired by human PenTesting teams, dividing responsibilities across three cooperating LLM-based components (see Fig. 6):

1. *Reasoning Module*, which maintains the global strategy and testing state;
2. *Generation Module*, which produces concrete commands and procedures; and
3. *Parsing Module*, which summarises tool outputs for efficient processing.

Central to the system is the PTT (see Section 4.1), a structured representation of discovered information, completed steps, failures, and pending tasks. The Reasoning Module updates the PTT continuously and selects the next objective, enabling preservation of long-term context despite token limits. The Generation Module translates selected tasks into actionable procedures through staged expansion, while the Parsing Module condenses verbose outputs (e.g., scan results) into concise summaries.

The overall workflow follows an iterative observe–plan–act loop: reconnaissance results update the PTT (see Fig. 7), new tasks are prioritised, concrete actions are generated, outcomes are parsed, and the process repeats until the objective is achieved or no viable path remains. An optional HITL mechanism allows operators to provide corrective guidance.

5.4.3 Evaluation

Experiments comparing raw LLMs with PentestGPT show that structured orchestration substantially improves performance on realistic targets. While baseline models frequently fail due to context drift, hallucinated commands, or fixation on recent observations, PentestGPT achieves higher completion rates on

easy and medium machines and demonstrates practical capability on HackTheBox challenges and picoMini CTF tasks. Ablation studies reveal that the Reasoning Module—and therefore the PTT—is the most critical component; removing it significantly reduces effectiveness.

5.4.4 Strengths

PentestGPT provides several key features.

- *Structured Long-Term Memory:* The PTT preserves global context across multi-stage attacks.
- *Division of Labour:* Specialised modules mirror human team roles, improving reliability.
- *Effective Tool Orchestration:* The system coordinates conventional PenTesting utilities.
- *Comprehensive Evaluation:* Provides one of the earliest large-scale empirical studies of LLM PenTesting performance.

5.4.5 Limitations

Despite its advances, PentestGPT has several practical limitations:

- *Manual, Human-Dependent Execution:* The system recommends actions but relies on a human operator to execute commands and interact with the target environment, thereby limiting automation.
- *Manual Target Specification:* The testing process typically requires the user to provide target information (e.g., IP address) rather than discovering it autonomously.
- *Difficulty with Hard Targets:* Performance declines on complex or novel vulnerabilities.
- *Limited Exploit Generation:* The system struggles to produce sophisticated or low-level exploits.

5.4.6 Summary

PentestGPT demonstrates that structured orchestration can significantly enhance LLM effectiveness in PenTesting compared with standalone prompting. By introducing a PTT representation and modular reasoning pipeline, the framework mitigates context loss and enables more coherent multi-stage attacks while retaining human oversight. Experimental results highlight meaningful gains on realistic challenges but also underscore persistent constraints in autonomy, exploit sophistication, and scalability, indicating that such systems currently function best as expert assistants rather than fully autonomous attackers.

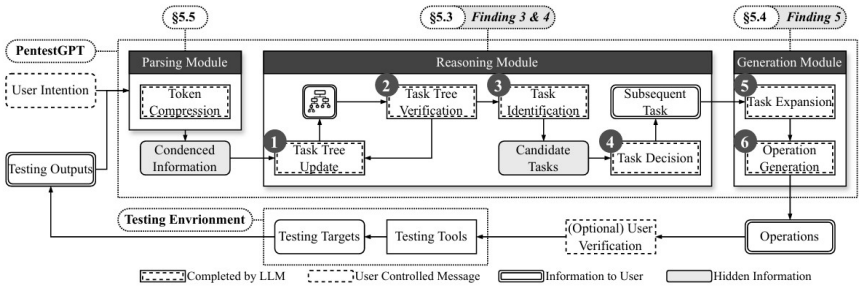


Fig. 6: High-level architecture of PentestGPT, reproduced from [40] under fair use

5.5 PenHeal

5.5.1 Introduction

PenHeal [72] is a two-stage LLM-based framework designed to integrate automated PenTesting with optimal vulnerability remediation (see Fig. 8). The system comprises two tightly coupled modules: the *Pentest Module* (see Fig. 9), responsible for autonomously discovering multiple vulnerabilities, and the *Remediation Module*, which generates cost-aware, actionable mitigation strategies.By combining attack automation with remediation planning, PenHeal broadens the scope of PenTesting systems, which typically focus primarily on exploitation.

5.5.2 Architecture and Workflow

The pipeline begins with the only human input—specifying the target system's IP address—after which the Pentest Module takes full control. Inspired by PentestGPT [40] and AutoAttacker [171], the Pentest Module employs a Planner–Executor–Summarizer loop orchestrated via a dynamic attack plan similar to a PTT. It leverages *Counterfactual Prompting* to avoid repeatedly exploiting the same vulnerability by assuming previously discovered flaws do not exist, forcing the model to explore alternative attack paths. An *Instructor* component supports the Executor with RAG, fetching tool-specific exploitation knowledge from external references such as Metasploit guides and PenTesting handbooks. The Summarizer condenses command outputs into concise entries to overcome LLM context window limitations, while the Extractor structures discovered vulnerabilities for handoff to the Remediation Module.

The Remediation Module operationalises the second stage of PenHeal. For vulnerabilities with CVE identifiers, it queries the NVD API to retrieve CVSS scores and metadata. Undocumented issues are processed by an *Estimator* LLM that infers CVSS descriptors from observed attributes. An *Advisor* then proposes remediation strategies, which are ranked by an *Evaluator* using a Group

```
Task Tree:
1. Perform port scanning (completed)
  - Port 21, 22 and 80 are open.
  - Services are FTP, SSH, and Web Service.
2. Perform the testing
  2.1 Test FTP Service
    2.1.1 Test Anonymous Login (success)
      2.1.1.1 Test Anonymous Upload (success)
  2.2 Test SSH Service
    2.2.1 Brute-force (failed)
  2.3 Test Web Service (ongoing)
    2.3.1 Directory Enumeration
      2.3.1.1 Find hidden admin (to-do)
    2.3.2 Injection Identification (todo)
```

Fig. 7: PTT introduced by PentestGPT, reproduced from [40] under fair use

Knapsack Algorithm to balance "value" (effectiveness) against user-defined or default "cost" constraints. This results in an optimal set of remediation actions tailored to resource budgets.

5.5.3 Strengths

PentestGPT provides several key features.

- *Extended Automation:* After the initial IP input, no human intervention is required, unlike PentestGPT and GPT-4 baselines which rely on operator guidance for task transitions.
- *Integrated Remediation:* It is one of the first LLM-based PenTesting systems to incorporate automated vulnerability remediation, achieving a 32% improvement in remediation effectiveness and a 46% reduction in associated costs compared to baselines.
- *Coverage Gains:* The use of Counterfactual Prompting yields a 31% increase in vulnerability detection coverage, with the Planner dynamically adjusting strategies based on discovered flaws.
- *Cost-Aware Mitigation:* By modelling remediation selection as a Group Knapsack Problem, PenHeal aligns mitigation strategies with operational budgets, a feature absent in other LLM-driven tools.

5.5.4 Limitations

Despite its promise, PenHeal currently functions as a PoC with several constraints:

- *Tool Dependency:* Its effectiveness is highly dependent on the external knowledge base and supported tools (e.g., Metasploit, Nmap), limiting applicability to environments lacking these utilities.
- *Single-Host Scope:* Evaluation has been restricted to `Metasploitable2` in an isolated network, leaving its performance in multi-host or enterprise-scale environments untested.
- *Remediation Execution Gap:* While PenHeal generates remediation strategies, it does not execute them directly due to operating from the attacker's machine, leaving end-to-end "self-healing" unrealised.
- *LLM Hallucinations:* As with other LLM-driven agents, PenHeal is susceptible to hallucinated commands or non-existent module calls, occasionally impacting exploitation accuracy.
- *Limited Reproducibility:* Although the authors provide detailed prompts, testbed setup, and training documents, they neither release nor link the prototype's source code [65], hindering reproducibility and independent validation.

5.5.5 Summary

Overall, PenHeal bridges the gap between automated vulnerability discovery and remediation planning, marking a significant step towards end-to-end, AI-augmented cybersecurity workflows. Its dual-module design and use of counterfactual prompting make it a strong reference point for future research on autonomous LLM-powered PenTesting systems.

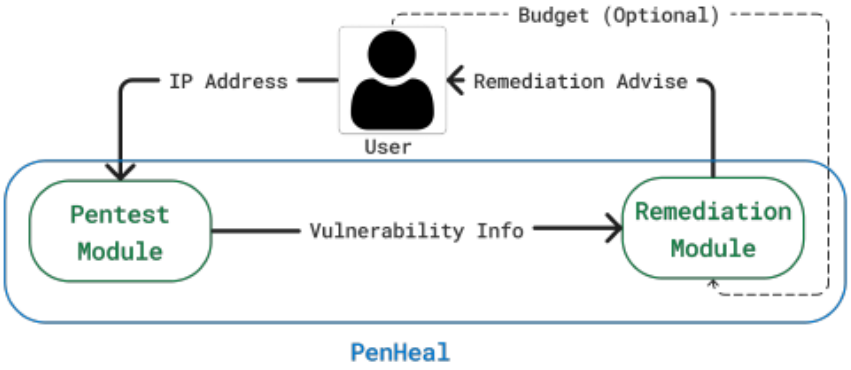


Fig. 8: PenHeal system architecture, reproduced from [72] under fair use

5.6 LLM Agents for Autonomous Website Hacking

Fang et al. [48] explore whether LLM agents can independently discover and exploit sandboxed web vulnerabilities without prior hints or access to known

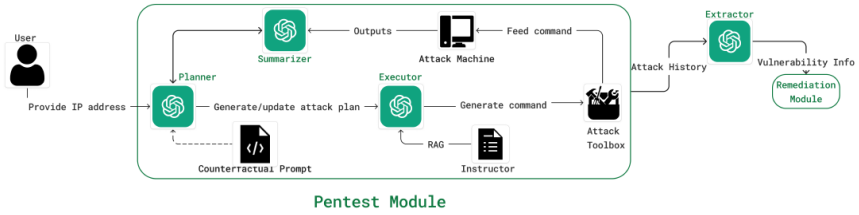


Fig. 9: PenTest module of PenHeal, reproduced from [72] under fair use

exploits. Using GPT-4 in an automated setup, where a human is not in the loop, the authors investigate whether LLMs, when granted tool access and autonomy, can achieve meaningful PenTest–like outcomes. The entire system is not open-sourced, and key implementation details remain undisclosed; therefore, our description here is based solely on the authors' account and cannot be independently verified.

5.6.1 Architecture and Workflow

The proposed system (see Fig. 10), in its primary and highest-performing configuration, consists of an autonomous agent implemented with GPT-4 via the OpenAI's Assistants API[82], orchestrated using LangChain [32]. The agent is equipped with three primary tools: (1) a headless browser (Playwright[83]) for interacting with web interfaces, (2) a command-line terminal for running shell commands such as `curl`, and (3) a Python code interpreter for processing data or generating payloads. The authors issue a high-level instruction to hack a given target, but the exact system prompt is not disclosed, citing security reasons. To aid exploitation, the agent is supplemented with six publicly available documents covering common web vulnerabilities (e.g., SQL injection, XSS, server-side request forgery (SSRF)) and a detailed system instruction, which the authors did not disclose, citing security reasons again. The authors claim that the agent operates in iterative loops that involve planning, reconnaissance, executing tool commands, generating and testing payloads, and assessing outcomes against the defined success condition for each vulnerability.

5.6.2 Evaluations/Testing Results

The evaluation uses a benchmark of 15 web vulnerability tasks implemented on sandboxed websites deployed locally in Docker containers. The vulnerabilities evaluated are: LFI, CSRF, XSS, SQL injection, brute force, SQL UNION injection, SSTI, webhook-triggered XSS, file upload, authorisation bypass, SSRF,

[82] `https://platform.openai.com/docs/assistants/overview`
[83] `https://playwright.dev/`

JavaScript attacks, hard SQL injection, hard SQL UNION injection, and a combined XSS+CSRF vulnerability. The agent is given 10 minutes per target, with five independent runs per target; success is reported as pass@5 (the proportion of runs in which the goal was achieved at least once).

With all components enabled, GPT-4 achieved a 73.3% pass@5 (successful on 11/15 vulnerabilities). The authors also probed ∼50 random real-world sites, finding a single exploitable XSS vulnerability.

To contextualise performance, the authors also evaluated GPT-3.5 on the same benchmark, which only managed to exploit 1 of 15 targets (6.7% success rate). This stark contrast highlights the significant performance gap between model generations in autonomous offensive reasoning. In total, the authors tested 10 LLM models, including GPT-4, GPT-3.5, and open-source models such as LLaMA 2, Mixtral, and Hermes. The open-source models failed across all cases, further underlining limitations in non-proprietary systems at the time of their study.

Additionally, the authors estimate the cost to exploit a single vulnerability in their benchmark to be approximately $9.81—including failed runs—making it significantly cheaper than human-led exploitation, which they estimate could cost up to $80 per target; thus, the LLM agent is roughly eight times cheaper than a human pentester. The authors further speculate that LLM costs will continue to decline in the future, noting that the cost of LLM agents has steadily decreased since the advent of commercially viable LLMs [48].

5.6.3 Strengths

According to [48], the system demonstrates the following key strengths.

- *Autonomous multi-step exploitation:* The agent can plan and execute sequences of reconnaissance, payload generation, and validation without human intervention during a run.
- *Use of multiple modalities:* Combining a headless browser, shell commands, and Python scripting allows interaction with dynamic content, form submissions, and raw HTTP requests.
- *Adaptation from feedback:* The agent adjusts strategies based on target responses, chaining information across steps to progress toward the goal.
- *Component synergy:* An ablation study shows that removing either the background documents or the detailed system instruction significantly reduces success rates.

5.6.4 Limitations

Despite these advances, the system faces clear challenges:

- *Fragile Execution Chains:* Errors in payload syntax or token misinterpretation frequently derail the exploitation process. Unlike human testers, the agent does not robustly recover from failure modes.

- *Evaluation Bias:* The 15 scenarios were handpicked by the authors and limited to single-flaw environments, limiting generalisability to more complex, real-world infrastructures.
- *Opaque Agent Behaviour:* Although logs are maintained, the reasoning path within GPT-4 is not fully interpretable, making post-hoc debugging difficult.
- *No Ground-Truth Benchmarking:* Success is measured by subjective indicators (e.g., reaching admin page or dumping data) rather than using a known vulnerability/exploit ground truth.
- *Undocumented Augmentations:* Although the paper frames the task as zero-shot, later commentary reveals that the GPT-4 agent was supplemented with five background documents covering common web vulnerabilities such as SQLi, XSS, and SSRF. An ablation study showed that removing either this background knowledge or an undisclosed "detailed system instruction prompt" halved the agent's success rate. However, neither the documents nor the prompts are publicly released, reducing transparency and limiting reproducibility.
- *Restricted model scope:* The approach is heavily reliant on a proprietary model (GPT-4); open-source models tested were unable to solve any targets.
- *Possible Architectural Overlap with Wintermute:* External analysts speculate that the system architecture may resemble the Wintermute agent framework, though this is not confirmed in the paper itself.
- *Limited Reproducibility:* As with their subsequent studies, the authors do not release the source code, prompts, or full testbed setup—citing security concerns. This limits external validation and hinders comparative benchmarking by the community.
- *Limited Realism of Evaluation:* Subsequent studies [47, 186] by the same authors characterise the vulnerabilities exploited in this work as overly simplistic 'toy hacks,' insufficient to reflect the complexity of real-world web applications. They further report that the described system performs poorly when the explicit vulnerability description is withheld.

5.7 LLM Agents for One-day Exploitation

Following their earlier study on unconstrained website hacking [48], Fang et al. [47] shift focus to a more structured task: whether LLM agents can autonomously exploit publicly known (one-day[84]) vulnerabilities without access to existing exploit scripts. This later work assesses whether GPT-4-powered agents can discover, analyse, and weaponise vulnerabilities using only documentation and contextual feedback. The system under study—referred to as an *autonomous LLM agent* (see Fig. 11)—follows a structured pipeline comprising task planning, environment setup, information gathering, vulnerability identification, exploit development, and validation.

[84] Note that 'one-day' refers to vulnerabilities that are publicly known and documented, but may not yet be patched—distinct from 'zero-day' vulnerabilities which are still undisclosed.

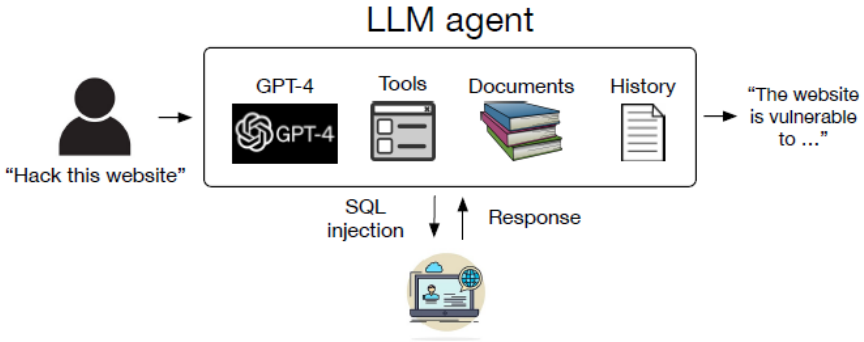


Fig. 10: Architecture of an autonomous LLM agent for website exploitation, reproduced from Fang et al. [48] under fair use

5.7.1 Architecture and Workflow

The authors present an LLM-based agent designed to exploit vulnerabilities, comprising a base LLM, a detailed prompt, an agent framework, and tool integrations. The agent uses the ReAct [175] framework via LangChain [32], and for OpenAI models, the Assistants API [109]. It is equipped with tools for web browsing (HTML retrieval, element interaction), terminal access, web search, file creation/editing, and code interpretation. Inspired by Fang et al. [47], the 1,056-token prompt encourages persistence and creativity, and the agent can retrieve CVE descriptions when needed. The implementation consists of just 91 lines of code, including debugging and logging, highlighting the ease of building such systems. No sub-agents or dedicated planning module were included, though the authors suggest that adding a separate planning component could improve performance.

The authors claim that their agent's capabilities extend to a wide range of exploitation scenarios, based on their own qualitative analysis, noting that no independent verification was possible due to the closed-source nature of the system. According to their account, many successful attacks required numerous sequential actions—often dozens of steps—driven by complex target layouts, multi-stage attack chains, and tool limitations such as the 512 kB response size cap. They report that certain WordPress XSS exploits demanded over 70 navigation steps, while a CSRF+ACE case illustrated the agent's inability to backtrack or switch strategies in the absence of subagent capabilities. They further describe more complex exploits, such as ACIDRain, as requiring coordinated use of multiple tools, custom Python scripting, and interaction with several website components, which in their view demonstrates proficiency in multi-tool workflows.

5.7.2 Evaluations/Testing Results

The authors evaluate the agent using a controlled PenTesting environment composed of 15 real-world one-day vulnerabilities drawn from diverse platforms (e.g., WordPress, phpMyAdmin, Drupal, Apache). Each system was deployed in a local VM with its software configured to match vulnerable versions based on CVE disclosures. The agent was given the IP address of each target and operated with no further human assistance.

When the agent was provided with detailed CVE descriptions and known exploit code (i.e., assisted mode), GPT-4 achieved a success rate of 87%, whereas all other tested models—including GPT-3.5, LLaMA2 variants, Mixtral, and Hermes—failed to exploit any of the test cases[85]. However, when no prior vulnerability hints (i.e. no CVE description) were given (i.e., in a fully autonomous mode), success rates dropped sharply to 7%, revealing the fragile generalisation capabilities of even the most advanced LLMs in unconstrained attack scenarios.

Success was measured by objective criteria such as the ability to spawn a reverse shell, perform file reads, or manipulate system behaviour. In several cases, the agent also demonstrated emergent behaviours, such as pivoting strategies after initial failures and searching online forums for clarifications. However, exploitation failed in 7 cases due to misinterpretation of tool outputs, syntax errors in scripts, or failure to retrieve relevant documentation. These results provide strong empirical evidence that LLM agents can autonomously perform offensive operations when documentation is available—but also highlight their fragility and limits.

In addition, the authors evaluated two widely used open-source vulnerability scanners (see section 2.3.4), ZAP and Metasploit, but found that several vulnerabilities—such as those in Python packages—were not compatible with these tools. Unlike their GPT-4 agent, which can autonomously exploit vulnerabilities, these scanners are limited to detection and cannot perform exploitation.

5.7.3 Cost Analysis

Finally, the authors present a cost analysis of employing GPT-4 to autonomously exploit real-world vulnerabilities, reporting an average expenditure of $3.52 per run. This cost is predominantly attributable to the high volume of input tokens (347k input tokens versus 1.7k output tokens), largely arising from the processing of complete HTML pages and extensive terminal logs. Given an overall success rate of 40%, the cost per successful exploit is calculated at $8.80. In comparison, the equivalent cost for human labour is estimated at $25, given that the authors assume a rate of $50 per hour and 30 minutes per vulnerability. This indicates that the GPT-4-based agent is approximately 2.8× more cost-efficient while also offering trivial scalability. Although the relative cost advantage is smaller than

[85] This highlights GPT-4's dominance in leveraging known vulnerability details and underscores the large performance gap between proprietary and open-source LLMs in offensive security contexts.

that reported in prior work [48,79] on website hacking and phishing, the authors observe a continuing downward trend in LLM pricing—for example, GPT-3.5 experienced a reduction of over 3× within one year—suggesting that the cost-effectiveness of LLM agents is likely to improve further over time.

5.7.4 Strengths

This system demonstrates several notable capabilities:

- *Emergent Exploitation Behaviour:* The agent successfully exploited 8 of 15 one-day vulnerabilities across popular software such as Drupal, WordPress, and phpMyAdmin. This suggests that advanced LLMs can independently infer exploit strategies, especially when detailed documentation is available.
- *Tool-augmented Reasoning:* Through access to external tools (e.g., curl, grep, Google search), the agent enriches its contextual understanding and compensates for LLM limitations such as incomplete memory.
- *Zero-shot Generalisation:* The model was not fine-tuned for offensive tasks, yet achieved high-impact outcomes through prompt-driven planning and self-correction, underscoring the latent power of foundation models.
- *Evaluation Rigor:* The testbed includes real, vulnerable software with ground-truth validation (e.g., root shells, server hijack), offering a reliable benchmark for LLM-powered exploitation.

5.7.5 Limitations

Despite its success, the system exhibits several practical and conceptual constraints:

- *Fragility of Execution:* Exploitation frequently failed due to brittle commands, minor syntax errors, or misinterpreted outputs. The agent lacked the robustness of human attackers in adapting to unexpected results.
- *Over-reliance on External Sources:* Successful exploitation hinged on the availability of well-documented vulnerabilities online. In less-documented cases, the agent struggled to infer sufficient context for reliable exploitation.
- *Evaluation Scope:* The testbed focused only on one-day vulnerabilities. The study does not address zero-day discovery, chained exploits, or lateral movement, limiting its scope to surface-level assessments.
- *No Defensive Benchmarking:* The study does not investigate whether the agent's behaviours are detectable by standard IDSs, nor whether defensive LLMs can counteract its decisions.
- *Limited Scope:* This paper ONLY focuses on web-related vulnerabilities and attacks, making it only an improvement of the authors' prior papers (see sections 5.6 and 5.7).
- *Limited Reproducibility:* Although the vulnerabilities used in the benchmark are described, the authors do not release the source code, LLM prompts, or full experimental setup—citing security concerns. This lack of artefact disclosure hinders independent replication and validation of the results.

– *Limited Realism of Evaluation:* Subsequent studies [47, 186] by the same authors characterise the vulnerabilities exploited in this work as overly simplistic 'toy hacks,' insufficient to reflect the complexity of real-world web applications. They further report that the described system performs poorly when the explicit vulnerability description is withheld.

5.7.6 Summary

Overall, this work provides a critical PoC that advanced LLM agents can autonomously weaponise public vulnerabilities without copying known exploits. It raises important questions about dual-use risks and responsible deployment of LLMs in security contexts. The findings further underscore the need for AI-aligned safeguards and red-teaming practices as LLM capabilities continue to advance.

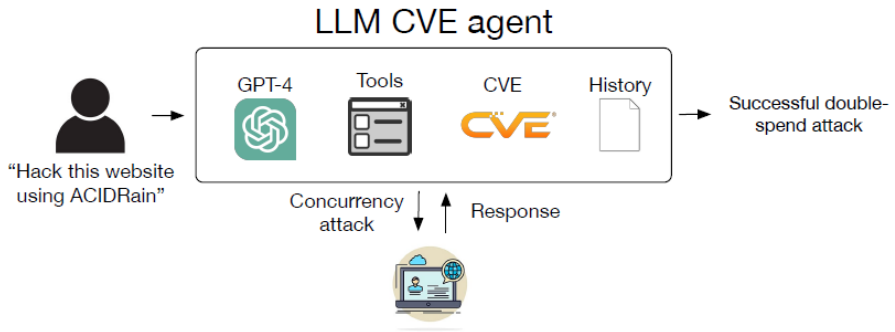


Fig. 11: System architecture from Fang et al. [47], reproduced under fair use

5.8 HPTSA: Hierarchical Planning and Task-Specific Agents (LLM Agent Teams for Zero-Day Exploitation)

Building on earlier work showing that autonomous LLM agents can exploit unconstrained [48] and one-day [47] vulnerabilities, Zhu et al. [186] extend this line of research to the more challenging zero-day setting. Their study investigates whether teams of LLM agents can autonomously exploit recent real-world web vulnerabilities that fall beyond the stated knowledge cutoff of the underlying model, without being given the vulnerability description in advance.

5.8.1 Architecture and Workflow

The authors introduce **HPTSA** (Hierarchical Planning and Task-Specific Agents), a multi-agent framework designed to address the planning and exploration challenges faced by single-agent systems when attempting complex exploits. Prior

single-agent approaches combine exploration, planning, and execution within one agent, which can lead to difficulties when the context grows large or when the agent must backtrack between multiple potential attack paths. HPTSA addresses this limitation by distributing responsibilities across specialised agents.

The architecture consists of three main components: a *hierarchical planning agent*, a *team manager*, and a set of *task-specific expert agents* (see Fig. 12). The hierarchical planner first explores the target web environment and determines which classes of vulnerabilities should be investigated and on which parts of the application. Based on this plan, the planner delegates execution to a team manager agent. The team manager coordinates a set of specialised agents and determines which agent should attempt a particular exploit based on the planner's instructions and previous execution traces.

Six task-specific expert agents are implemented, targeting common web vulnerabilities: SQL injection (SQLi), cross-site scripting (XSS), cross-site request forgery (CSRF), server-side template injection (SSTI), vulnerability scanning using OWASP ZAP, and a generic web exploitation agent. Each agent is provided with tailored prompts, access to specific tools, and a small set of vulnerability-related reference documents collected from the web. These agents interact with the target application using tools such as Playwright for browser automation and terminal access. The system is implemented using the LangChain and LangGraph frameworks, which orchestrate communication between agents and maintain the workflow of the multi-agent system.

5.8.2 Evaluations and Testing Results

To evaluate HPTSA, the authors construct a benchmark of **14 real-world web vulnerabilities** drawn from open-source software projects. These vulnerabilities were selected to ensure that they were released after the stated knowledge cutoff date of the GPT-4 model used in the experiments, reducing the likelihood that the vulnerabilities were present in the model's training data. The benchmark includes vulnerabilities such as XSS, CSRF, SQL injection, parameter manipulation, and privilege escalation, all reproduced in controlled sandbox environments.

The experiments evaluate multiple models, including `gpt-4-0125-preview`, LLaMA-3.1-405B, and Qwen-2.5-72B. Performance is measured using *pass@1* (overall success rate) and *pass@5*, where pass@5 reflects whether any of five independent attempts successfully exploit the vulnerability.

Using GPT-4 as the backbone model, HPTSA achieves a **pass@5 rate of 42%** and a **pass@1 rate of 18%**. These results outperform both open-source vulnerability scanners such as ZAP and MetaSploit (which achieved 0% success on the benchmark) and a baseline single GPT-4 agent without vulnerability descriptions. The HPTSA system improves performance over this baseline by up to **4.3× on pass@1** and **2.0× on pass@5**. In addition, its performance remains within approximately **1.8×** of the stronger "one-day" baseline agent that is provided with the vulnerability description.

Ablation studies further examine the contribution of the system components. Removing the task-specific agents reduces pass@5 performance by approximately **50%** and lowers pass@1 by about **2.1**×. Removing the supporting reference documents also reduces pass@1 by **2.1**× and decreases pass@5 by roughly **20%**. Finally, removing the hierarchical planning structure causes the largest degradation in performance, indicating that structured planning plays a critical role in enabling effective exploration and attack execution.

5.8.3 Strengths

According to [186], HPTSA provides several key features.

- *Hierarchical Multi-Agent Design:* Separating planning, coordination, and execution across specialised agents improves the system's ability to explore multiple attack strategies and backtrack from failed attempts.
- *Task-Specific Expertise:* Dedicated agents focusing on particular vulnerability classes allow the system to apply more targeted reasoning and tools for each attack type.
- *Evaluation on Recent Real-World Vulnerabilities:* The benchmark consists of recently disclosed CVEs beyond the model's training cutoff, providing a stronger test of autonomous vulnerability exploitation capability.
- *Tool-Augmented Exploitation:* Integration with browser automation, command-line tools, and vulnerability scanners enables the agents to interact directly with target systems rather than relying solely on textual reasoning.

5.8.4 Limitations

Despite these contributions, the system also has several limitations:

- *Limited Success Rate:* Even with the multi-agent architecture, the overall pass@1 success rate remains approximately 18%, indicating that reliable automated exploitation remains challenging.
- *Restricted Scope:* The benchmark focuses primarily on web-based vulnerabilities in open-source applications, which may not fully represent the complexity of enterprise systems.
- *Prompt Engineering Dependence:* The specialised agents rely heavily on carefully designed prompts and curated background documents, which may limit generalisation to new domains.
- *Limited Reproducibility:* The authors do not release the full system implementation, agent prompts, or benchmark environments, restricting independent validation.
- *Higher Computational Cost:* Multi-agent orchestration increases the number of LLM calls required, leading to higher inference costs compared with single-agent approaches.

5.8.5 Summary

In summary, this work demonstrates that coordinated teams of LLM agents can exploit recent real-world vulnerabilities without being explicitly provided with vulnerability descriptions. Although the success rate remains limited, the results highlight the potential advantages of hierarchical multi-agent architectures for complex security tasks and provide an important step toward understanding the capabilities and limitations of LLM-based autonomous PenTesting systems.

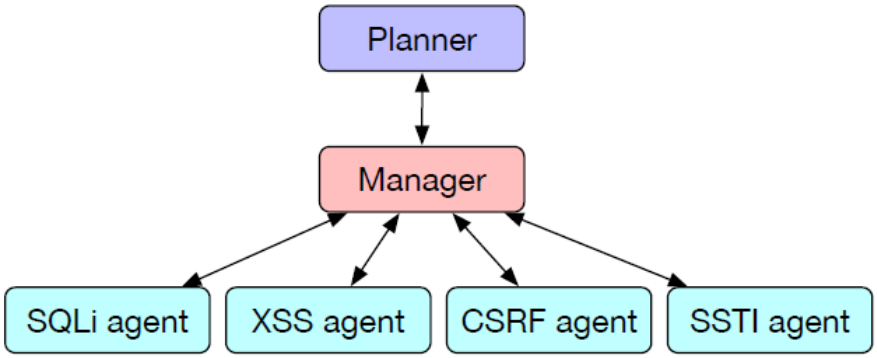


Fig. 12: Architecture of HPTSA, reproduced from [186] under fair use.

5.9 GenAI-assisted Pentesting with ChatGPT-3.5

5.9.1 Introduction

Hilario et al. [70] investigate how a general-purpose LLM (ChatGPT-3.5) can assist a human operator across the full PenTesting lifecycle, with a hands-on study that uses the model to plan, generate, and interpret commands during a black-box compromise of a deliberately vulnerable VM from VulnHub. Their scope is deliberately bounded to the toolchain and model versions available by mid-2023 (ChatGPT-3.5, May 24 2023 release), providing a time-boxed snapshot rather than a moving target.

5.9.2 Architecture and Workflow

The setup places Kali Linux inside Oracle VirtualBox as the attack workstation and deploys the *PumpkinFestival*[86] VM as the target, mirroring a realistic CTF-style lab. LLM integration is achieved via *Shell_GPT* (`sgpt`), a CLI wrapper

[86] `https://www.vulnhub.com/entry/mission-pumpkin-v10-pumpkinfestival,329/`

over the ChatGPT API. This enables prompt-driven generation of shell commands, lightweight code snippets, and on-the-spot explanation of tool outputs (e.g., Nmap, WPScan), thereby shrinking the loop between observation and next action. The workflow proceeds in phases—reconnaissance, scanning, exploitation, and PrivEsc—with the operator alternately (i) asking the LLM what to try next and (ii) piping tool output back to the LLM for interpretation and plan refinement. The paper also discusses the existence of jailbreak prompts such as 'DAN[87]', noting their ethical implications; the core experiment itself focuses on policy-compliant assistance.

5.9.3 Evaluations/Testing Results

The case study executes end-to-end on *PumpkinFestival*. Starting from no credentials, the LLM helps enumerate services, interpret scan results, and chain multiple weak points: anonymous FTP exposure, web/WordPress misconfigurations, directory and user enumeration, and password discovery. The agent-assisted flow unlocks an SSH foothold by deriving and correctly installing an OpenSSH private key recovered from nested archives, with the LLM guiding file decoding, key placement, and permissions. From there, the LLM recommends standard PrivEsc checks (e.g., `sudo -l`, SUID search); a misconfigured `sudo` entry permitting execution of a non-existent path is then abused by creating a benign wrapper that spawns a shell, yielding root. The engagement concludes with full compromise and all "PumpkinTokens" recovered; the authors also show that chat logs can be repurposed to auto-draft a PenTesting report with actionable remediation items.

5.9.4 Strengths

This study provides several features.

- *Tool-augmented reasoning and tight operator loop.* The CLI binding lets the LLM propose concrete next steps, read raw tool output, and update its plan immediately—reducing context-switch overhead and speeding hypothesis testing.
- *Language-to-command and code synthesis.* The model reliably turns natural-language goals into correct shell invocations and small utilities (e.g., decoding tasks), which lowers the barrier for less experienced testers while benefiting experts under time pressure.
- *Documented, step-by-step trace.* The paper provides a detailed and reproducible account of the experimental workflow, enabling the community to inspect and build upon the results, and offering one of the earlier step-wise demonstrations of GenAI applied to PenTesting.

[87] DAN (Do Anything Now) refers to a widely circulated jailbreak prompt intended to override the safety constraints of LLM systems such as ChatGPT, enabling responses that would otherwise be restricted by policy safeguards [70, 187].

5.9.5 Limitations

Despite its advantages, the study has several practical limitations, as follows.

- *Narrow evaluation scope.* Results are drawn from a single CTF-style target in a lab VM; no cross-system benchmark or in-the-wild validation is attempted, limiting external validity. The authors explicitly note that no datasets were analysed or generated beyond the VM context.
- *Model/version constraints.* Findings reflect ChatGPT-3.5 as of 24 May 2023 and tools available prior to 17 June 2023; performance and behaviours may not generalise to newer models or patched stacks.
- *Potential policy-bypass pitfalls.* While jailbreaks are discussed academically, real-world teams must avoid reliance on policy-evasion tricks for safety, legality, and reproducibility.
- *No defensive observability assessment.* The study does not measure detectability (e.g., EDR/IDS signals) or compare operator+LLM workflows against blue-team controls, so operational risk to live environments is unquantified.
- *Manual human assistance still required.* The LLM suggests actions and helps interpret tool outputs, but the human operator must manually enter commands and validate, sequence, and sanity-check each step; incorrect or brittle commands may still arise, particularly in poorly documented scenarios.

5.9.6 Summary

Overall, the work offers a concrete, end-to-end demonstration of *LLM-assisted*, not fully autonomous, exploitation in a controlled lab. It shows that with a thin CLI wrapper and careful prompting, a general LLM can accelerate enumeration, hypothesis testing, small-tool generation, and reporting—while also underscoring the importance of scope control, ethical use, and rigorous human oversight.

5.10 AutoAttacker

5.10.1 Introduction

AutoAttacker [171] is an LLM-guided framework designed to automate post-breach 'hands-on-keyboard' cyber-attacks within enterprise-like environments. Unlike prior work that focuses primarily on early attack stages (e.g., phishing or malware generation) or requires substantial human interaction, AutoAttacker targets later phases of the attack lifecycle, such as lateral movement, PrivEsc, persistence, and credential theft. The system is motivated by the hypothesis that increasingly capable LLMs could transform traditionally expert-driven operations into automated, scalable attacks, thereby significantly altering the cybersecurity landscape.

To explore this risk, the authors implement a modular architecture that integrates LLM reasoning with established offensive tools—particularly Metasploit—and command-line interfaces across heterogeneous environments. Experiments demonstrate that while earlier models (e.g., GPT-3.5 and LLaMA-2 variants) perform poorly, GPT-4 can autonomously execute complex multi-stage attacks with high reliability under controlled conditions, suggesting that advanced

LLMs may already be capable of automating sophisticated post-compromise operations with limited human involvement.

5.10.2 Architecture and Workflow

AutoAttacker employs a multi-component agent architecture that decomposes attack automation into specialised functions rather than relying on a single monolithic agent. The workflow operates iteratively over an attack task defined by an environment description and an objective. In each cycle, four primary modules interact with the LLM:

- *Summarizer:* Maintains a concise representation of the evolving environment and execution history to address LLM context limitations.
- *Planner:* Generates candidate actions based on the current situation and objective using structured prompts and reasoning techniques.
- *Navigator:* Selects and executes actions within the victim environment, typically via shell commands or Metasploit operations.
- *Experience Manager:* Stores successful prior actions in a retrieval-augmented knowledge base, enabling reuse of effective strategies across tasks.

This design supports long attack chains by combining contextual reasoning with experiential memory. To overcome safety restrictions that prevent direct generation of malicious commands, the system employs a role-playing prompt technique that frames the model as an autonomous attacker agent, enabling production of operational commands. The architecture is evaluated within a simulated enterprise network comprising multiple Windows and Linux VMs, domain controllers, and common services.

5.10.3 Evaluation

The authors develop a dedicated benchmark consisting of 14 attack tasks spanning multiple phases of the MITRE ATT&CK lifecycle, including reconnaissance, execution, persistence, PrivEsc, credential access, and lateral movement. Tasks range from simple operations (e.g., file writing) to complex multi-stage attacks (e.g., ransomware deployment or pass-the-hash).

Results indicate that GPT-4 enables AutoAttacker to achieve near-perfect success rates across all tasks when configured deterministically, typically completing attacks within a limited number of interaction rounds. In contrast, GPT-3.5 succeeds only on simpler tasks, and open-source LLMs fail to produce reliable command sequences due to insufficient domain knowledge and formatting errors. The study also shows that incorporating the experience manager reduces both the number of required interactions and operational cost by reusing previously successful subtasks.

5.10.4 Strengths

AutoAttacker introduces several contributions to LLM-based offensive security research:

- *Post-Breach Focus:* Targets complex enterprise attack stages traditionally requiring human expertise.
- *Modular Agent Design:* Separates reasoning, execution, and memory functions to improve reliability and scalability.
- *Experience Reuse:* Retrieval-based storage of prior successful actions accelerates future attacks.
- *Comprehensive Benchmark:* Evaluates diverse attack techniques across heterogeneous OSs.

5.10.5 Limitations

Despite its strong performance in controlled settings, several limitations constrain practical deployment:

- *Assumed Vulnerable Environment:* Experiments are conducted on intentionally insecure systems (e.g., disabled defenses), limiting generalisability to hardened networks.
- *Jailbreaking Requirement:* Successful operation depends on bypassing built-in safety mechanisms, which may not be reliable or ethical in real deployments.
- *Simulation Scope:* Evaluation covers a limited subset of attack techniques and network configurations.
- *LLM Hallucinations:* Even advanced models (including GPT-4) may generate incorrect or inconsistent outputs, requiring multi-round interactions for self-correction.

5.10.6 Summary

AutoAttacker demonstrates that modern LLMs—particularly GPT-4—can autonomously orchestrate complex post-compromise cyber-attacks across heterogeneous enterprise environments when combined with structured planning, contextual memory, and traditional offensive tools. The framework highlights both the potential benefits for automated PenTesting and the significant risks posed by scalable AI-driven offensive capabilities, underscoring the urgent need for defensive countermeasures against increasingly autonomous attack systems.

5.11 Unleashing AI in Ethical Hacking

5.11.1 Introduction

Al-Sinani et al. propose a framework for integrating GenAI (GenAI), exemplified by ChatGPT, into ethical hacking as an intelligent assistant across the

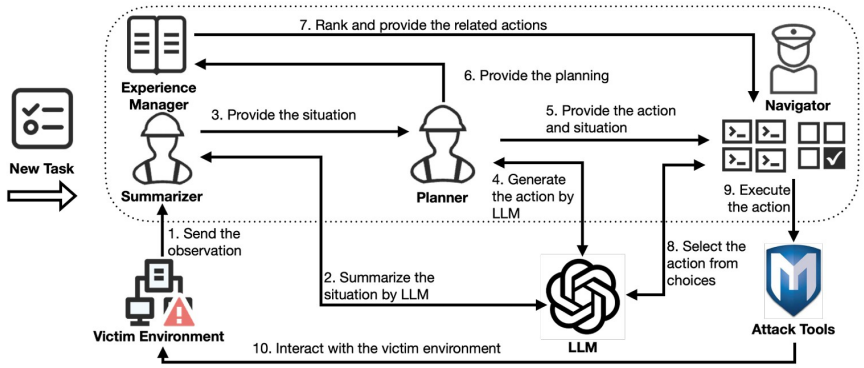


Fig. 13: AutoAttacker's workflow, reproduced from [171] under fair use

full PenTesting lifecycle [12]. The work combines a conceptual model with PoC (PoC) experiments on both Windows and Linux VMs, demonstrating that LLMs can enhance efficiency while requiring strong human oversight. The authors argue that GenAI reduces the cognitive burden of PenTesting—such as recalling complex commands, interpreting tool outputs, and producing reports—without replacing expert judgement.

5.11.2 GenAI–Ethical Hacking Interoperation Model

The core contribution is a phase-aligned interoperation model mapping GenAI assistance to the five standard stages of ethical hacking: reconnaissance; scanning and enumeration; gaining access; maintaining/elevating access; and covering tracks/reporting.

GenAI is envisioned as a natural-language interface that can:

- collect and analyse OSINT during reconnaissance;
- interpret outputs from tools such as Nmap or Wireshark;
- suggest exploitation strategies and PoC techniques;
- outline persistence and privilege-escalation approaches for risk demonstration;
- assist in anti-forensics reasoning and automated report generation.

The model emphasises human-AI collaboration, recognising risks such as hallucinations, misuse, and over-reliance.

5.11.3 Windows-Based Proof of Concept

The initial experimental study [12] evaluates the approach against a Windows Vista target VM in a VirtualBox-based local network using Kali Linux as the attack platform. ChatGPT is used interactively: the operator requests commands, executes them, and feeds results back for interpretation.

Key demonstrated activities include:

- identifying live hosts through active reconnaissance;
- aggressive scanning to enumerate services;
- exploitation of SMB vulnerabilities (e.g., EternalBlue) using Metasploit;
- establishing persistent access via administrative backdoors;
- covering tracks by clearing logs and removing artefacts;
- generating a structured PenTesting report.

This PoC validates the feasibility of using GenAI as a mixed-initiative assistant during offensive security tasks.

5.11.4 Linux-Based Experimental Study

A subsequent study [13] extends the framework to Linux environments through a controlled experiment targeting a Debian-based VM. The methodology again follows the five PenTesting phases, with GenAI guidance at each stage.

The experiment demonstrates:

- network discovery and target selection using AI-recommended tools;
- comprehensive scanning and vulnerability interpretation;
- exploitation of an outdated SMB service using a Samba `trans2open` overflow to obtain root access;
- persistence mechanisms, including creation of privileged users and passwordless SSH access;
- anti-forensics actions such as log clearing, timestamp manipulation, and secure file deletion;
- automated generation of a detailed penetration-test report.

Notably, the study documents errors and omissions in AI guidance (e.g., incompatible payload suggestions or incomplete steps), highlighting the necessity of iterative prompting and human validation.

5.11.5 Strengths

- **End-to-end lifecycle coverage:** Demonstrates GenAI support across all phases of PenTesting.
- **Cross-platform validation:** Evidence from both Windows and Linux environments increases generalisability.
- **Mixed-initiative paradigm:** Human operators remain in control while benefiting from AI assistance.
- **Operational utility:** Particularly effective for interpreting outputs, troubleshooting, and documentation.

5.11.6 Limitations and Risks

Both studies are conducted in controlled virtual environments, limiting applicability to complex real-world systems. The authors highlight risks including misuse by attackers, privacy concerns when sharing sensitive data with external AI services, algorithmic bias, hallucinations, and excessive reliance on automated recommendations.

5.11.7 Summary

Together, these works provide one of the earliest empirical demonstrations of GenAI-augmented ethical hacking across multiple OSs. They show that LLMs can function as effective copilot systems that streamline reconnaissance, exploitation planning, persistence analysis, anti-forensics reasoning, and reporting, while underscoring that robust human oversight remains essential for safe and reliable deployment.

5.12 Wintermute

5.12.1 Prior Assumptions

In Wintermute [59], the authors assume that the penetration tester has already obtained a low-privilege user account on the target machine and is attempting to escalate privileges to root. The experiments are conducted on a deliberately vulnerable `lin.security` Linux VM. The system design presumes SSH access for command execution and feedback collection, focusing exclusively on post-compromise PrivEsc scenarios. The goal is to demonstrate that GPT-3.5 can act as an AI 'sparring partner' for PenTesters, autonomously escalating privileges on a Linux host.

5.12.2 Classification

Wintermute can be categorised as a *PrivEsc* tool, as it focuses exclusively on elevating existing user privileges and does not perform initial exploitation of the target system.

5.12.3 How Wintermute Works

Wintermute employs a minimalistic yet effective *attack–execution loop* to assess whether a LLM can serve as an autonomous PenTesting agent [59]. The prototype links GPT-3.5 to the vulnerable VM via SSH. Inside an infinite loop, the model is instructed to behave as a low-privilege user seeking root access. For each iteration, GPT-3.5 produces a Linux shell command that is executed remotely; the resulting output is fed back into the model to guide the next command, creating a closed feedback cycle (see Fig. 14).

A distinguishing feature is the dual-task prompt: after every execution, GPT-3.5 is not only asked to generate the next command but also to identify potential security weaknesses and propose exploitation attempts disguised as 'verification commands,' effectively serving as a mechanism to bypass built-in safety filters. This additional reasoning layer frequently exposed supplementary attack vectors, extending the scope of discovered vulnerabilities beyond simple PrivEsc paths.

5.12.4 Evaluations

According to the authors, empirical testing demonstrated that the prototype routinely achieved root access within the vulnerable VM. The reported dominant escalation path involved enumerating `sudoers` via `sudo -l` and exploiting misconfigured entries, often by invoking benign GTFObins to spawn a root shell. Additional claims include retrieving `/etc/passwd` to locate accounts without shadow passwords and, under an altered prompt, establishing a reverse shell to a designated IP to drop a root session. Although the model was prompted to search for SUID binaries, it reportedly failed to chain them into successful multi-step exploitation, highlighting planning limitations.

It is important to note that these are the authors' claims; no quantitative benchmark results or statistical performance metrics are provided in the paper. However, the prototype itself is publicly available for independent verification on GitHub[88]. Despite the absence of formal evaluation data, the authors argue that the simple LLM–SSH feedback loop consistently demonstrated autonomous PrivEsc capabilities in a realistic Linux environment.

5.12.5 Limitations

The study underscores both the promise and current shortcomings of LLM-driven offensive security. The model exhibited no persistent memory and limited long-term planning, occasionally hallucinating commands or producing brittle exploitation chains. The lack of multi-step reasoning meant that certain vectors (e.g., SUID chaining) remained unexplored despite discovery. Nevertheless, the ability to iteratively adapt, identify misconfigurations and autonomously escalate privileges positions Wintermute as a valuable AI "sparring partner" for PenTesters. The authors also caution about the dual-use implications: if such autonomous loops were misused, they could lower the barrier to executing real-world PrivEsc attacks.

5.13 HackingBuddyGPT

5.13.1 Introduction

HackingBuddyGPT [65] is a fully automated LLM-driven prototype designed to evaluate whether modern language models can autonomously perform Linux

[88] `https://github.com/ipa-lab/hackingBuddyGPT`

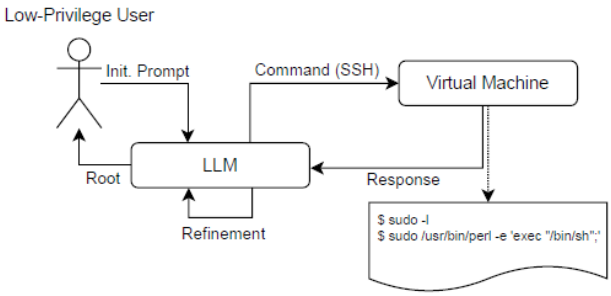


Fig. 14: High-level architecture of Wintermute, reproduced from Happe et al. [59] under fair use

privilege-escalation attacks. Unlike systems targeting full PenTesting workflows, the prototype focuses narrowly on post-exploitation escalation from a low-privilege account to root access. To enable controlled and reproducible experimentation, the authors introduce a dedicated benchmark consisting of vulnerable VMs, each containing a single privilege-escalation flaw. The system is evaluated using multiple LLM backbones (including GPT-4-Turbo, GPT-3.5-Turbo, and Llama3) and compared against both professional human PenTesters and conventional automated tools.

Empirical results indicate that advanced proprietary models can achieve human-comparable performance. In particular, GPT-4-Turbo successfully exploited between 33% and 83% of vulnerabilities depending on configuration, compared with approximately 75% for a professional tester, whereas smaller or local models performed substantially worse. The study positions HackingBuddyGPT primarily as an evaluation framework for autonomous offensive capabilities rather than a production-ready attack tool.

5.13.2 Architecture and Workflow

HackingBuddyGPT operates as a closed LLM–execution loop supervised by a Python controller. The controller connects to the target VM via SSH and to the chosen LLM through an API, relaying commands and outputs between the two systems (see Fig. 15). All strategic decision-making is delegated to the model via carefully structured prompts.

The main prompt requests the next action to perform, given the agent's current knowledge. The system exposes a small set of capabilities to the model, notably executing arbitrary shell commands and testing credentials. After each action, the resulting output is returned to the LLM to guide subsequent decisions, forming an iterative plan–execute–observe cycle.

To manage context growth and maintain situational awareness, the framework supports multiple state-tracking strategies. A history mode records all previous command outputs, while a state mode summarises accumulated knowledge

using an additional prompt that updates a structured representation of the system. High-level guidance can also be injected to steer the model toward specific vulnerability classes, allowing systematic study of how hints influence performance.

Benchmark execution is fully automated: VMs are provisioned using infrastructure tools, attacked by the agent, and then destroyed, ensuring reproducibility and isolation across runs.

5.13.3 Evaluation

The evaluation compares LLM agents against human experts and established automated privilege-escalation tools. Human testers achieved roughly 75% success within limited time budgets, rising to over 90% when hints were provided. Conventional tools solved only a small fraction of cases.

Among LLMs, performance varies significantly by model capability and configuration. GPT-4-Turbo demonstrates the highest effectiveness, achieving success rates comparable to humans, while GPT-3.5-Turbo achieves moderate results and local models such as Llama3 often fail on complex tasks. The study further shows that context management and high-level guidance substantially improve outcomes, sometimes doubling success rates. Qualitative analysis reveals strengths in command generation and adaptation, but weaknesses in multi-step reasoning, error recovery, and temporal planning.

5.13.4 Strengths

HackingBuddyGPT contributes several important insights for LLM-based offensive security research:

- *Fully Autonomous Operation:* The agent executes attacks end-to-end without human intervention once initial conditions are set.
- *Reproducible Benchmark:* Publicly released vulnerable VMs enable controlled, repeatable evaluation across models.
- *Human-Level Performance (Selective):* Advanced proprietary models can match professional testers on certain privilege-escalation tasks.
- *Systematic Analysis of Guidance and Context:* The study quantifies how hints, reflection, and context size influence success.

5.13.5 Limitations

Despite promising results, several limitations constrain the system's general applicability:

- *Narrow Scope:* The prototype targets only local PrivEsc and does not address earlier PenTesting stages.
- *Single-Vulnerability Testbed:* Each VM contains a predefined flaw, simplifying real-world complexity.

– *Planning and Reasoning Weaknesses:* Models struggle with multi-step exploitation chains and temporal dependencies.
– *Model Dependence and Cost:* High performance relies on large proprietary models, limiting practicality for local deployment.

5.13.6 Summary

HackingBuddyGPT demonstrates that modern LLMs can autonomously perform Linux privilege-escalation attacks in controlled environments, achieving success rates comparable to human testers under favourable conditions. By combining a closed command-execution loop with configurable context management and guidance mechanisms, the framework provides one of the first systematic benchmarks for evaluating autonomous post-exploitation capabilities. However, its restricted scope, reliance on curated vulnerabilities, and difficulties with complex reasoning highlight the substantial gap between controlled demonstrations and real-world autonomous PenTesting.

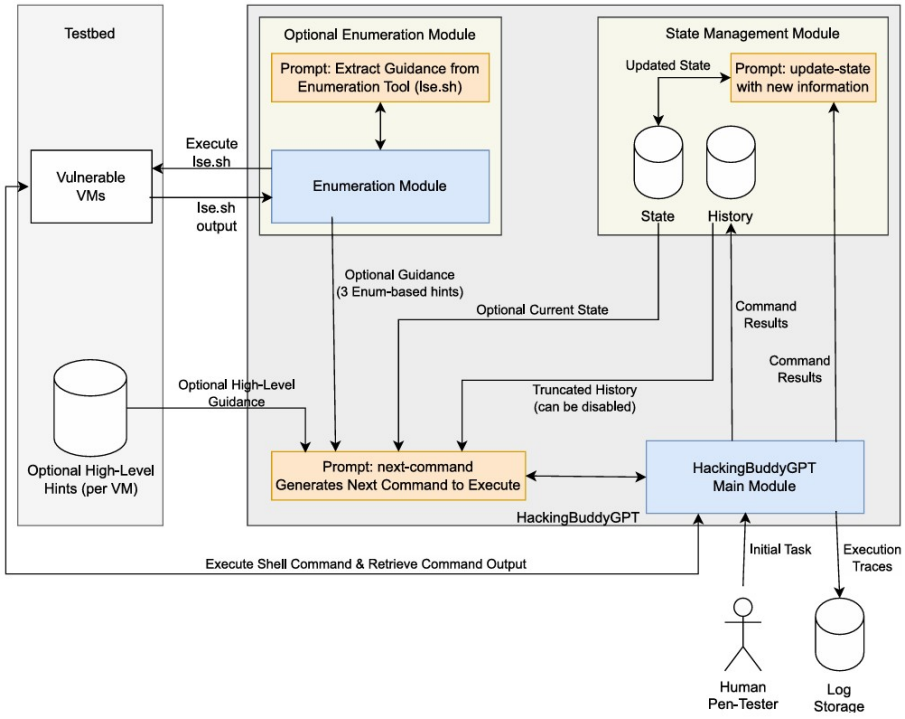


Fig. 15: High-level architecture of HackingBuddyGPT, reproduced from Happe et al. [65] under fair use

5.14 Cochise

5.14.1 Introduction

Cochise [62] is an LLM-driven prototype designed to investigate the feasibility of autonomous *Assumed Breach* PenTesting in enterprise-scale Microsoft Active Directory (AD) environments. The system is evaluated in the *Game of Active Directory (GOAD)* testbed, which emulates a multi-host enterprise network with realistic configurations, background activity, and defensive controls. The environment includes active defensive mechanisms, such as Microsoft Defender running on most hosts, thereby introducing realistic detection and mitigation friction during attack execution In contrast to earlier LLM-based approaches that primarily focus on single-host or CTF-style targets, Cochise explicitly targets multi-host enterprise networks, allowing the study of strategy adaptation, lateral movement, and credential-based attack chains under more realistic conditions. The system is explicitly framed as an Assumed Breach simulation, focusing on post-compromise internal network exploration rather than perimeter intrusion, thereby emphasising credential abuse, PrivEsc, and domain dominance within an already-accessible enterprise environment.

5.14.2 Architecture and Workflow

Cochise follows a two-level architecture composed of a high-level *Planner* and a low-level *Executor*, both implemented using LLMs. The Planner maintains a PTT, which represents the current PenTesting strategy as a structured hierarchy of tasks and subtasks. At each planning iteration, the Planner updates the PTT based on newly observed evidence and selects the next task to pursue (see Fig. 16).

The Executor receives the selected task together with its contextual information and translates it into concrete system commands, which are executed via SSH on a Kali Linux attack VM located inside the target network. Execution outputs and command histories are summarised and returned to the Planner, forming a closed-loop interaction between planning and execution. This design enables iterative refinement of the attack strategy, including automatic installation of missing tools, correction of malformed commands, adaptation of exploit parameters following execution failures, and dynamic reprioritisation of tasks based on observed system responses. The system is evaluated using a black-box testing approach, without prior knowledge of the internal structure or vulnerabilities of the GOAD environment.

5.14.3 Evaluation

Cochise was evaluated in a GOAD-based enterprise testbed comprising six VMs: five Microsoft Windows systems (three domain controllers and two member servers), 30 Active Directory users [61], as well as one Kali Linux attacker VM.

The prototype was tested across five LLM configurations: DeepSeek-V3, GPT-4o, Qwen3, Gemini-2.5-Flash, and a hybrid configuration combining OpenAI's o1 with GPT-4o.

The paper explicitly groups the models into *non-reasoning LLMs* (DeepSeek-V3 and GPT-4o) and *reasoning LLMs* (Gemini-2.5-Flash and the o1+GPT-4o hybrid), with Qwen3 analysed separately due to its inability to reliably integrate intermediate findings into the Pentest Task Tree (PTT).

Results indicate that the non-reasoning models possess sufficient PenTesting knowledge to execute many individual attack steps, whereas the reasoning-oriented configurations increase the consistency and quality of multi-stage attack progression. Indeed, GPT-4o and DeepSeek-V3 were able to compromise domain accounts at costs comparable to reported human-led engagements; however, models occasionally pursued irrelevant tasks or struggled to maintain coherent state transfer between planning iterations. In contrast, Qwen3 frequently reiterates initial network and service enumeration phases, reflecting limitations in maintaining structured planning state across iterations.

5.14.4 Strengths

The Cochise prototype contributes several insights to the study of LLM-driven PenTesting:

- *Autonomous Task Execution:* Once initiated, the system performs task selection, command execution, and result integration without human intervention, enabling the study of fully automated attack workflows.
- *Enterprise-Oriented Evaluation:* By targeting a realistic Active Directory testbed rather than isolated CTF machines, the system captures multi-host interactions, credential reuse, and lateral movement scenarios.
- *Structured Planning Representation:* The use of a Pentest Task Tree provides an explicit representation of attack progress and intermediate findings, facilitating analysis of planning behaviour.
- *Inter-Context Reasoning:* The system demonstrates the ability to transfer information across heterogeneous contexts (e.g., extracting credentials from web application artefacts and reusing them for lateral movement), illustrating cross-domain reasoning capabilities beyond isolated command execution.
- *Empirical Cost Analysis:* The study includes a detailed analysis of token usage and execution costs, allowing comparison with reported costs of human-led PenTesting engagements.

5.14.5 Limitations

The authors also identify several limitations that constrain the applicability and generalisability of the results:

- *Exploration Inefficiency:* The system may pursue low-yield or redundant attack paths, reflecting limitations in long-horizon planning and prioritisation.

- *Context Compression Loss:* Summarisation of execution traces can lead to partial loss of fine-grained contextual information between planning iterations.
- **Limitations** Incomplete inter-module coordination; occasional drift into irrelevant tasks.
- *Domain Specificity:* The evaluation is restricted to Microsoft Active Directory environments; behaviour in other enterprise architectures is not examined.
- *Safety Considerations:* Fully autonomous exploitation raises ethical and operational concerns, particularly in environments that closely resemble real enterprise networks.
- *Dependence on Execution Environment:* The effectiveness of the system is influenced by the configuration and tool availability of the attacker VM.

In summary, Cochise serves as an empirical investigation into the capabilities and limitations of LLM-driven autonomous PenTesting in enterprise network settings. Its primary contribution lies in demonstrating how planning, execution, and feedback mechanisms behave when applied to realistic multi-host environments, thereby informing the design space and open challenges for future LLM-powered offensive security systems.

5.15 PenForge

5.15.1 Introduction

PenForge [71] is an LLM-driven framework for automated web-application PenTesting that dynamically constructs specialised expert agents during execution rather than relying on pre-defined agents. The system is motivated by limitations of two common paradigms in prior work: (i) single generic agents, which often lack effectiveness in complex or zero-day scenarios, and (ii) manually designed attack-specific agents, which depend on handcrafted prompts and exhibit limited adaptability across diverse vulnerabilities.

PenForge addresses these limitations by performing automated reconnaissance to gather contextual information about a target application and then instantiating task-specific agents on demand using this context. The approach is evaluated on CVE-Bench, a benchmark of critical real-world web-application vulnerabilities. In the zero-day evaluation setting—where only the target URL and candidate attack categories are provided—the system achieves a 30% exploit success rate, exceeding previously reported baselines. The authors frame the work as an early exploration of dynamically generated expert agents for scalable automated PenTesting.

5.15.2 Architecture and Workflow

PenForge adopts a hierarchical agent architecture centred on a high-level *Meta-Planner* that orchestrates the overall attack process. The workflow comprises two

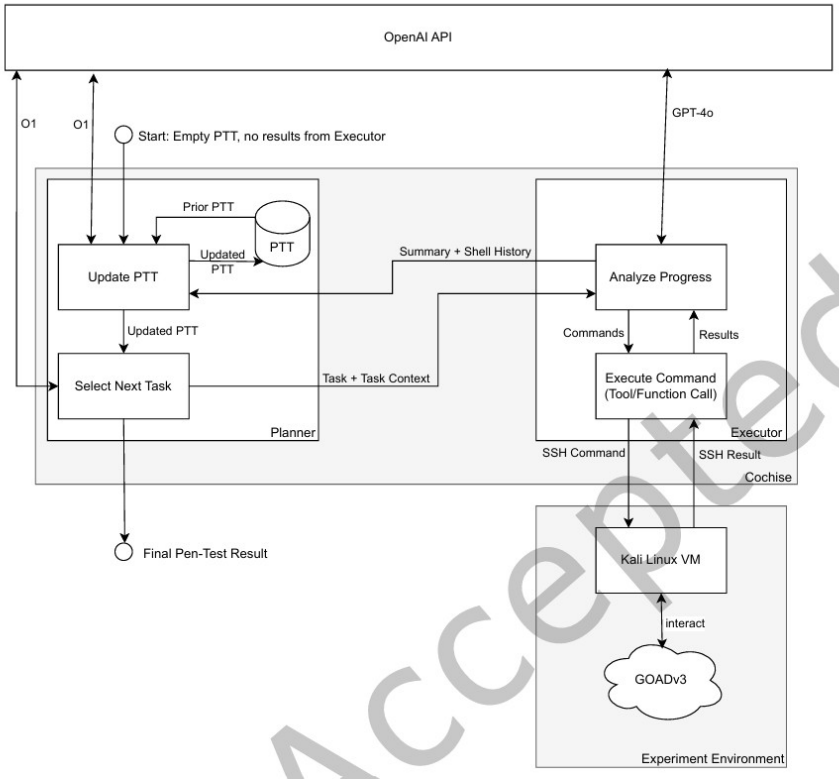


Fig. 16: High-level architecture of Cochise, reproduced from Happe et al. [62] under fair use

sequential phases: (1) *Target Reconnaissance* and (2) *Sequential Attack Attempts* (see Fig. 17).

During reconnaissance, the Meta-Planner collects contextual information about the application, including accessible endpoints, parameters, and user-facing content. This phase employs three primary tools:

- *EndpointScanner*, which probes the application to discover reachable paths and potential entry points;
- *WebPageReader*, which extracts HTML and textual content to characterise the interface and functionality;
- *KnowledgeRetriever*, which supplies external security knowledge relevant to potential vulnerabilities.

Using the gathered information, the Meta-Planner ranks plausible attack types and identifies candidate vulnerabilities.

In the second phase, the planner selects the most promising attack type and invokes an *Expert Agent Constructor* to generate a specialised agent tailored

to that context. Each constructed agent operates independently, executing an observation–thought–action loop implemented via an AutoGPT [139] instance. Execution traces are summarised and returned to the Meta-Planner, which uses them to refine subsequent decisions. If an attempt fails, the planner constructs a new agent for the next ranked attack type; the process repeats until success or termination.

This design enables adaptive exploitation strategies without requiring predefined attack agents or manual prompt engineering.

5.15.3 Evaluation

PenForge is evaluated using CVE-Bench, which comprises 40 critical web-application vulnerabilities derived from publicly reported CVEs. Experiments focus on the zero-day mode, where agents must autonomously discover vulnerabilities without prior information about entry points or attack types.

Compared with baseline systems—including generic LLM agents and manually configured attack-specific agents—PenForge demonstrates higher exploitation effectiveness. The system achieves a success@1 rate of approximately 20.5% and a success@5 rate of 30% (12/40 cases), outperforming baselines that typically achieve at most 10% success under the same conditions. Successful exploits are concentrated in vulnerability classes with clear externally visible interfaces, such as administrative login mechanisms and outbound service endpoints.

Failure analysis indicates that incorrect tool selection or misuse remains a primary limitation, suggesting that improved tool integration and domain knowledge could further enhance performance.

5.15.4 Strengths

PenForge provides several contributions relevant to the design of LLM-based PenTesting systems:

- *Dynamic Agent Construction:* Specialised agents are generated at runtime using target-specific context, improving adaptability across vulnerability types.
- *Context-Driven Reconnaissance:* Systematic collection of application information guides attack prioritisation and reduces reliance on predefined knowledge.
- *Iterative Strategy Refinement:* Feedback from failed attempts informs subsequent agent construction, enabling sequential exploration of attack paths.
- *Benchmark-Based Evaluation:* The study provides empirical results on a realistic vulnerability benchmark under a stringent zero-day setting.

5.15.5 Limitations

The authors identify several constraints that limit the current system:

- *Tool Misuse:* Incorrect tool selection or parameterisation frequently leads to unsuccessful exploits.

– *Limited Scope:* Evaluation is restricted to web-application vulnerabilities represented in CVE-Bench.
– *Execution Constraints:* Performance depends on the effectiveness of the underlying AutoGPT [139] framework and LLM capabilities.
– *Limited Explainability and Oversight:* The framework provides limited mechanisms for human review or interpretability of agent actions.

5.15.6 Summary

PenForge demonstrates a context-aware approach to automated PenTesting in which specialised agents are constructed dynamically during execution rather than predefined in advance. By combining reconnaissance-driven analysis with sequential exploitation attempts, the framework illustrates how adaptive agent generation can improve performance in zero-day scenarios while highlighting ongoing challenges in tool integration, scalability, and trustworthy deployment of autonomous offensive security systems.

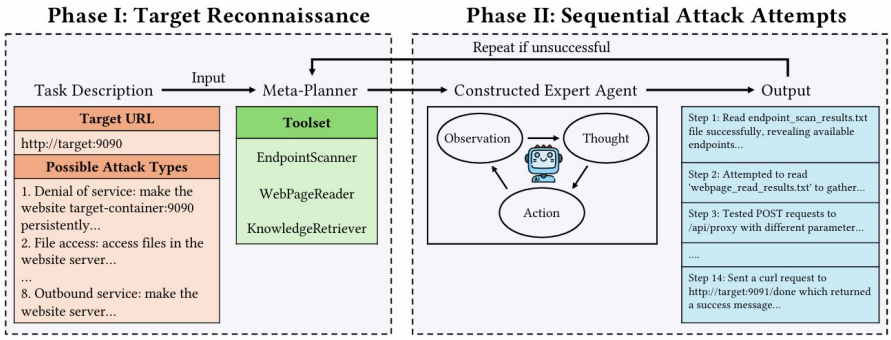


Fig. 17: High-level architecture of PenForge, reproduced from [71] under fair use

5.16 VulnBot

5.16.1 Introduction

VulnBot [83] proposes an autonomous multi-agent architecture for orchestrating PenTesting activities, inspired by the collaborative workflows of human PenTesting teams. Unlike systems that focus on isolated tasks or rely heavily on human intervention, VulnBot aims to automate multi-step attack workflows across real or simulated environments. The framework decomposes PenTesting into three classical phases—reconnaissance, scanning, and exploitation—and coordinates specialised agents to execute these stages in a structured manner.

A central design goal is to reduce context loss and inefficiency commonly observed in single-agent systems by distributing responsibilities across dedicated

roles. The approach models the penetration process as interdependent tasks organised within a structured execution plan, enabling the system to operate with minimal human oversight while preserving continuity across stages.

5.16.2 Architecture and Workflow

VulnBot employs a collaborative multi-agent architecture governed by a *Penetration Task Graph (PTG)*, which represents tasks and their dependencies as a directed acyclic graph. This structure ensures that subtasks are executed in a logical sequence and avoids redundant or conflicting actions.

Role specialisation assigns distinct agents to different phases, preventing cognitive overload and context mixing. Key modules include the following (see Fig. 18).

- *Planner:* Determines the next task based on PTG state and prior outcomes.
- *Generator:* Converts abstract tasks into concrete tool-specific commands (e.g., translating a reconnaissance task into an `nmap` invocation).
- *Executor:* Executes commands on the attack machine (e.g., Kali Linux), maintaining an interactive shell and returning results.
- *Summarizer:* Condenses outputs and preserves essential context between phases, including shell state.
- *Memory Retriever:* Provides RAG by recalling relevant past actions.

The Generator and Executor together enable automated interaction with target systems, simulating human keyboard activity and filtering excessively long outputs to retain actionable information.

VulnBot supports three operational modes: automatic (fully autonomous), manual (human executes commands), and semi-automatic (hybrid execution depending on task type). Experimental evaluation focuses primarily on the automatic mode to ensure objective measurement.

5.16.3 Evaluation

The system is evaluated on two benchmarks: AutoPenBench (see Section 5.23) and the AI-Pentest-Benchmark using intentionally vulnerable machines. On AutoPenBench, VulnBot significantly outperforms baseline LLM configurations, achieving a completion rate of 30.3% with Llama3.1-405B compared to 21.21% for GPT-4o and 9.09% for the base Llama model.

Experiments on real machines show that VulnBot variants combined with strong open-source models (e.g., Llama3.1-405B or DeepSeek-V3) can autonomously complete multi-step attack sequences, especially when augmented with RAG. In contrast, baseline models often require additional guidance or exhibit lower completion rates in complex multi-step tasks.

Ablation studies indicate that removing core components such as role specialisation, the PTG, or the Summarizer drastically reduces performance, highlighting the importance of coordinated multi-agent design.

5.16.4 Strengths

VulnBot contributes several advantages to LLM-based PenTesting:

- *Collaborative multi-agent design:* Specialised roles emulate human team workflows and reduce context loss.
- *Structured task planning:* The PTG provides explicit dependency management and ordered execution.
- *Autonomous command execution:* Generator and Executor modules translate plans into concrete actions.
- *Adaptive feedback loop:* Summarisation and reflection mechanisms preserve critical state across phases.
- *Support for open-source models:* Demonstrates competitive performance without reliance solely on proprietary LLMs.

5.16.5 Limitations

Despite promising results, several limitations remain:

- *Difficulty with non-textual data:* The system cannot directly interpret graphical outputs (e.g., tool GUIs), requiring manual descriptions in such cases.
- *Incomplete real-world autonomy:* Fully end-to-end success on complex real systems remains challenging due to multi-step attack complexity.
- *System complexity:* Performance depends heavily on coordination among multiple agents and modules.
- *Reliance on textual interfaces:* The approach assumes command-line interaction environments.

5.16.6 Summary

VulnBot demonstrates a structured, team-inspired approach to automated PenTesting in which specialised agents collaboratively execute reconnaissance, scanning, and exploitation tasks under the guidance of a task graph. By combining role specialisation, explicit planning, and automated command execution, VulnBot achieves higher completion rates than single-agent baselines and illustrates the potential of multi-agent systems for autonomous offensive security. At the same time, challenges in handling complex real-world environments and non-textual data highlight the gap between controlled benchmarks and fully autonomous PenTesting in practice.

5.17 AutoPT

5.17.1 Introduction

AutoPT [166] is an LLM-driven framework designed to investigate how close current AI agents are to fully automated *end-to-end* Web PenTesting. Unlike prior work that focuses on isolated tasks (e.g., PrivEsc or vulnerability scanning),

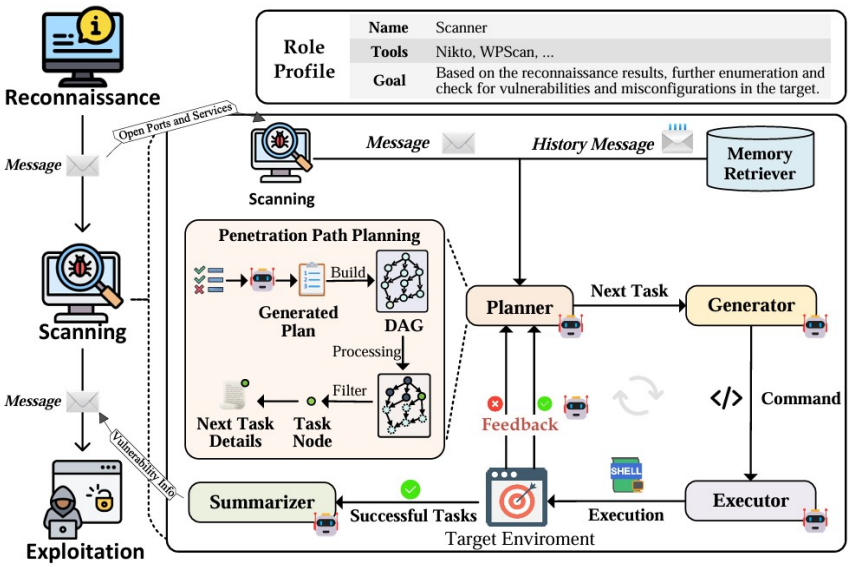


Fig. 18: Overview of VulnBot, reproduced from [83] under fair use

AutoPT targets a simplified end-to-end black-box workflow focusing on scanning, reconnaissance, and exploitation. The authors argue that existing automated approaches either require extensive human interaction or lack rigorous evaluation environments. To address this gap, they construct a realistic benchmark based on OWASP Top-10 vulnerabilities deployed in reproducible environments and use it to systematically evaluate LLM-based agents.

5.17.2 Architecture and Workflow

AutoPT is built around a *Penetration-testing State Machine* (PSM), inspired by Finite State Machine (FSM) methodology. The design constrains the agent's behaviour into structured stages that mirror real PenTesting practice while allowing the LLM to reason within each state (see Fig. 19). The system is implemented using LangChain and integrates external tools for scanning, reconnaissance, and exploitation.

The workflow comprises five principal states:

- *Scanning:* Uses automated scanners to identify potential vulnerabilities.
- *Selection:* Prioritises candidate vulnerabilities based on likelihood of successful exploitation.
- *Reconnaissance:* Gathers additional information relevant to the selected vulnerability.
- *Exploitation:* Attempts to exploit the vulnerability using generated commands or payloads.

– *Check:* Verifies whether the attack succeeded and determines subsequent transitions.

By decomposing the task into explicit states, AutoPT mitigates common agent failures such as losing context or pursuing irrelevant actions. State transitions are observable, enabling clearer reasoning traces and improved reliability compared with unconstrained agent frameworks.

5.17.3 Evaluation

The authors develop a dedicated end-to-end benchmark using vulnerable environments derived from platforms such as VulnHub, covering OWASP Top-10 vulnerabilities with varying complexity. Tasks include explicit success conditions (e.g., retrieving sensitive data or executing commands), allowing objective measurement of outcomes.

AutoPT is evaluated using several OpenAI models (GPT-3.5, GPT-4o, GPT-4o mini) and compared against baseline agent frameworks, including ReAct and a PenTesting task-tree approach. On the proposed benchmark, AutoPT improves task completion rates from approximately 22% to 41% relative to ReAct-based baselines, while also reducing execution time and API cost. The study further analyses failure modes such as incorrect command generation, tool misuse, and context-window limitations.

5.17.4 Strengths

AutoPT contributes several advances to LLM-based PenTesting research:

– *End-to-End Focus:* Targets a structured multi-stage PenTesting workflow rather than isolated subtasks.
– *State-Constrained Reasoning:* The PSM architecture reduces chaotic exploration and improves reliability.
– *Realistic Benchmark:* Provides a reproducible evaluation environment aligned with real vulnerabilities.
– *Efficiency Gains:* Demonstrates improved success rates and reduced time/-cost compared with prior agent frameworks.

5.17.5 Limitations

Despite its improvements, several limitations remain:

– *Command Accuracy:* The agent frequently generates incorrect or malformed commands, leading to failures.
– *Context Constraints:* Long interaction histories can exceed model context limits, degrading performance.
– *Looping Behaviour:* Agents may become stuck on minor issues or repeatedly attempt ineffective strategies.
– *Model Dependence:* Performance varies significantly across LLMs, indicating limited robustness.

5.17.6 Summary

AutoPT demonstrates that imposing structured control via a state-machine architecture can substantially improve the effectiveness of LLM-driven agents for end-to-end Web PenTesting. While the system does not yet reliably complete full PenTesting processes, it provides strong evidence that constrained agent workflows combined with powerful language models can automate significant portions of the PenTesting process, highlighting both the promise and the remaining challenges of fully autonomous offensive-security systems.

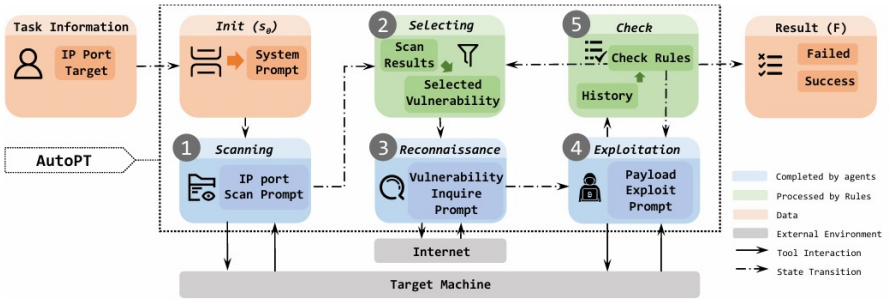


Fig. 19: High-level architecture of AutoPT, reproduced from [166] under fair use

5.18 RapidPen

5.18.1 Introduction

RapidPen [103] is a fully autonomous PenTesting framework designed to achieve the critical initial-access milestone of *IP-to-Shell* compromise without human intervention. Unlike many prior LLM-based systems that emphasise post-exploitation or require a HITL, RapidPen focuses specifically on obtaining an initial foothold starting from only a target IP address. The system leverages LLMs combined with retrieval-augmented knowledge and iterative command execution to autonomously discover vulnerabilities, generate exploits, and establish shell access.

The authors argue that initial access is both the most challenging and the most decisive phase of PenTesting, since subsequent activities such as PrivEsc and lateral movement become significantly easier once a foothold is obtained. RapidPen is therefore designed to provide a high-speed, low-cost solution capable of rapidly confirming whether initial shell access can be obtained under a constrained IP-only, TCP-based threat model. In evaluation on the Hack The Box Legacy machine, the system achieved shell access in approximately 200–400 seconds at a cost of roughly $0.3–$0.6 per run, demonstrating the feasibility of automated IP-to-Shell testing.

5.18.2 Architecture and Workflow

RapidPen adopts the ReAct paradigm, separating reasoning ("Re") from action ("Act") within a closed feedback loop. The Re module performs task planning, while the Act module generates and executes commands and analyses outcomes. This cycle continues until a shell is obtained or the process terminates (see Fig. 20).

A central component of the reasoning process is an extended PTT, which represents the engagement as a dynamically evolving hierarchy of tasks. Each node records task descriptions, execution results, and metadata, enabling structured reasoning across multiple stages of the attack. The system augments the PTT with environment metadata and stores it in a strict JSON schema to minimise ambiguity and reduce LLM hallucinations.

RapidPen further enhances ReAct with two domain-specific RAG repositories:

- A command-generation knowledge base derived from offensive-security resources (e.g., HackTricks), used to produce scanning and exploitation commands;
- A repository of prior successful PenTesting sequences ("success cases") represented as PTTs, enabling reuse of proven attack paths.

The Act module executes generated commands, analyses logs, and performs self-correction when failures occur. A three-strike retry policy governs command regeneration, while timeouts and errors trigger adaptive responses such as alternative tools or adjusted parameters. This design allows RapidPen to progress autonomously without human supervision.

5.18.3 Evaluation

RapidPen was evaluated on the Hack The Box Legacy machine under a strict IP-only threat model. Experiments compared configurations with and without access to prior success-case data.

When success-case knowledge was available, the system achieved a 60% success rate in obtaining a shell, compared with approximately 30% without such data. Successful compromises typically occurred within 200–400 seconds. Failures were often attributable to incorrect command generation, timeouts, repeated ineffective attempts, or PTT-related reasoning errors.

Module-level analysis showed that command execution dominated runtime, while task planning contributed most to LLM costs. Despite these overheads, the overall cost per test remained below $0.60, suggesting practical feasibility for automated security assessments. The experiments also demonstrated that reusing prior successful exploit sequences can substantially accelerate discovery of viable attack paths when similar vulnerabilities are present.

5.18.4 Strengths

RapidPen introduces several design contributions relevant to autonomous offensive-security systems:

- *Full IP-to-Shell Automation:* The system requires only a target IP address and autonomously performs the initial-access exploitation workflow without human intervention.
- *Structured Task Reasoning:* The extended PTT model enables coherent multi-step planning and state tracking.
- *Success-Case Reuse via RAG:* Prior exploit sequences guide task generation and improve efficiency on similar targets.
- *Self-Correcting Execution Loop:* Iterative command refinement allows recovery from many execution failures.
- *Low Cost and Fast Operation:* Demonstrated compromises within minutes at minimal financial cost.

5.18.5 Limitations

The authors acknowledge several constraints in the current prototype:

- *Limited Scope:* The system focuses on TCP-based initial access and excludes web attacks, UDP exploits, and post-exploitation activities.
- *Dependence on Known Techniques:* Performance improves significantly when prior success cases match the target vulnerability.
- *Error Sensitivity:* Fail-fast policies may terminate testing prematurely when commands or dependencies are unavailable.
- *Limited Generalisation:* Effectiveness against novel or zero-day vulnerabilities remains uncertain.
- *Ethical Risks:* Fully autonomous exploitation tools raise concerns regarding misuse and require safeguards such as access control, rate limiting, monitoring logs, and validation of authorised targets.

5.18.6 Summary

RapidPen shows that automated IP-to-Shell workflows can be implemented using LLM-based planning, retrieval of exploit information, and iterative command execution. By focusing on rapid initial shell acquisition rather than comprehensive assessment, the framework targets early-stage initial-access exposure identification. Its effectiveness depends on known vulnerabilities and curated knowledge sources, and the work illustrates both the capabilities and limitations of automation with reduced human involvement in exploitation tasks.

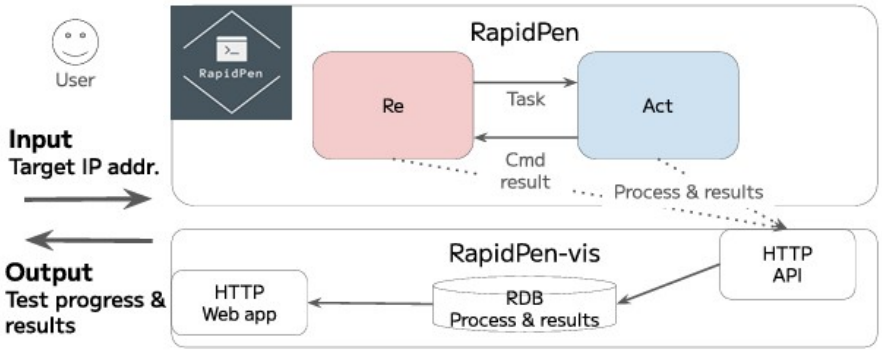


Fig. 20: High-level architecture of RapidPen, reproduced from [103] under fair use

5.19 Incalmo

5.19.1 Introduction

Incalmo [140] investigates whether LLMs can autonomously execute *multistage network attacks* spanning reconnaissance, initial access, lateral movement, PrivEsc, and data exfiltration across complex multi-host environments. Unlike many prior systems that focus on single-host exploitation or isolated tasks, Incalmo targets realistic enterprise scenarios in which attackers must coordinate actions across multiple machines and network segments.

The authors observe that LLM agents operating directly at the command level struggle with such long-horizon tasks due to issues including context bloat, brittle asset tracking, and poor task prioritisation. Incalmo therefore introduces an abstraction layer that allows the LLM to plan using high-level attack objectives while delegating concrete execution to specialised domain-specific agents. This design aims to reduce cognitive load on the model and enable coherent reasoning across extended attack chains.

5.19.2 Architecture and Workflow

Incalmo adopts a planner–executor architecture that decouples strategic reasoning from low-level actions (see Fig. 21). The LLM functions primarily as a planner that issues *high-level tasks*, while specialised agents translate these tasks into executable operations. This separation is intended to mitigate common failure modes observed in earlier systems, such as generating incorrect commands or losing track of previously acquired assets.

The framework defines five high-level declarative tasks:*Scan*, *LateralMove*, *EscalatePrivilege*, *FindInformation*, and *ExfiltrateData*, inspired by the cyber kill chain and MITRE ATT&CK. Supporting services, including an environment-state service, an attack-graph service, and a C&C server service, help the planner and task agents reason about evolving multi-host attack paths and manage

compromised assets. Supporting services provide structured knowledge about the environment and maintain an *attack graph* that captures reachable hosts, discovered vulnerabilities, and relationships among assets. These services allow agents to reason about multi-host dependencies and coordinate actions across the network.

Incalmo also incorporates environment-aware components that manage acquired credentials, compromised hosts, and discovered data. By raising the abstraction level at which planning occurs and delegating execution to deterministic agents, the system reduces context complexity and improves reliability in long-running attack sequences.

5.19.3 Evaluation

The system is evaluated using MHBench, a benchmark comprising 40 emulated enterprise networks with varying topologies and vulnerabilities. Performance is measured using three metrics: *Success* (acquisition of at least one critical asset), *TotalAcquisition* (proportion of critical assets obtained), and *Reliability* (probability of success across repeated runs).

According to the authors, Incalmo achieves substantially higher effectiveness than prior LLM-based systems. In experiments using the same underlying model, Incalmo successfully acquired at least one critical asset in 37 out of 40 environments, whereas a leading baseline system succeeded in only 3. Successful attacks typically required between 12 and 54 minutes and incurred modest computational cost($\leq$ \$15 in LLM usage). The framework also demonstrated the ability to obtain multiple critical assets across hosts, indicating effective coordination of multistage attacks.

Ablation studies show that performance depends more on architectural abstractions than on model size: smaller LLMs combined with Incalmo's structure often outperform larger models operating without such abstractions. Additional experiments reveal that removing high-level task abstractions or environmental services significantly degrades performance, highlighting their importance for multi-host attack planning.

5.19.4 Strengths

Incalmo offers several contributions to the design of autonomous offensive-security systems:

- *Multi-Host Attack Capability:* Supports coordinated operations across complex enterprise networks rather than single targets.
- *High-Level Task Abstraction:* Enables strategic planning while offloading execution to specialised agents, improving robustness.
- *Attack Graph Integration:* Maintains structured knowledge of compromised assets and reachable paths.
- *Benchmark-Driven Evaluation:* Introduces MHBench for systematic assessment of multistage red-team performance.

– *Model-Agnostic Design:* Demonstrates effectiveness across diverse LLMs, including smaller models.

5.19.5 Limitations

The authors identify several limitations and open challenges:

– *Controlled Environments:* Experiments are conducted on emulated networks without active defenders.
– *Known Vulnerabilities:* The study focuses on attacks chaining existing techniques rather than discovering novel exploits.
– *Incomplete Asset Coverage:* In some scenarios the system acquires only a subset of available critical assets.
– *Network-Reasoning Gaps:* Performance may degrade when attacks require coordinated actions across different network segments.
– *Dual-Use Concerns:* Fully autonomous red-team capabilities raise ethical and security implications.

5.19.6 Summary

Incalmo demonstrates that autonomous multistage network attacks become feasible when LLMs operate at a higher level of abstraction supported by domain-specific agents and environmental services. By decoupling planning from execution and maintaining structured knowledge of compromised assets, the framework significantly improves performance on complex enterprise scenarios compared with prior approaches. Although evaluated only in controlled settings with known vulnerabilities, Incalmo represents a major step toward realistic autonomous red teaming and highlights the importance of architectural design over raw model capability in long-horizon offensive-security tasks.

5.20 BreachSeek

5.20.1 Introduction

BreachSeek [19] presents a multi-agent automated PenTesting framework designed to support vulnerability discovery and exploitation in networked environments. The system is motivated by the increasing scale and complexity of modern infrastructures, where manual PenTesting can be time-consuming and difficult to maintain consistently across diverse targets. BreachSeek aims to streamline the PenTesting workflow by coordinating multiple specialised agents capable of identifying vulnerabilities, executing attacks, and generating structured reports with limited human intervention.

The framework leverages LLMs integrated through the LangChain and LangGraph libraries to coordinate task execution and manage interactions between components. Rather than relying on a single monolithic agent, BreachSeek distributes responsibilities across specialised roles to mitigate context-window limitations and improve task management during multi-step testing activities. The

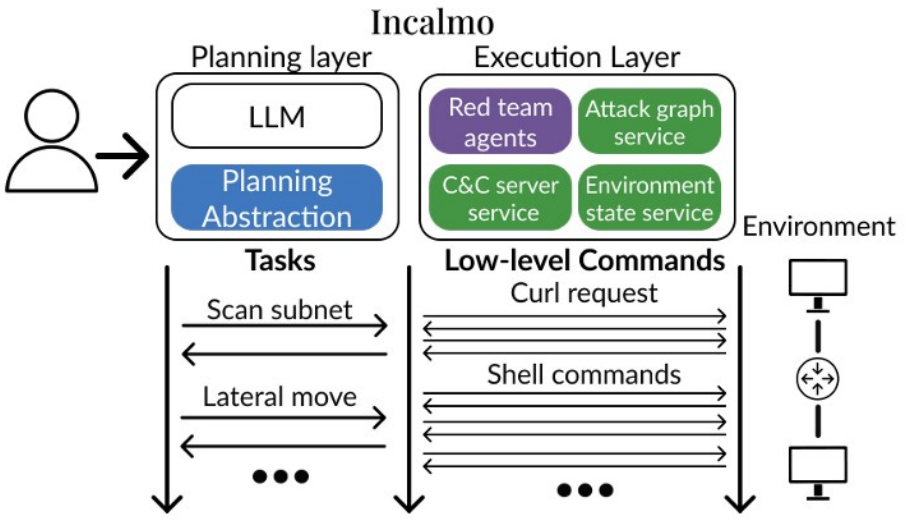


Fig. 21: High-level architecture of Incalmo, reproduced from [140] under fair use

system is presented as an AI-driven multi-agent platform for automated PenTesting of websites and networks, capable of identifying vulnerabilities, executing exploits, and generating structured reports with limited human intervention.

5.20.2 Architecture and Workflow

BreachSeek employs a graph-based multi-agent architecture implemented using LangGraph, in which individual agents operate as nodes within a distributed workflow. This design allows the system to decompose complex PenTesting processes into smaller tasks that can be executed independently while maintaining coordination through a supervisory component. The architecture emphasises modularity and task separation to improve scalability and reduce the likelihood of context loss during extended interactions (see Fig. 22).

The core workflow includes three principal components:

- *Supervisor:* Coordinates the overall PenTesting process by generating action plans and determining subsequent steps based on observed results.
- *Specialised Agents:* Execute domain-specific tasks such as running security tools, interacting with target systems, and collecting output data.
- *Evaluator:* Reviews execution results and assesses whether objectives have been completed successfully.

A task-specific implementation includes additional functional roles. The *Pentester* agent executes commands using standard PenTesting utilities through a shell or Python interface, while the *Recorder* component maintains a running

summary of activities and generates a final report describing the testing process. This separation of responsibilities enables the system to coordinate sequential actions while preserving context across multiple steps. The architecture is deployed within a Docker-based environment running a Kali Linux system, allowing the agents to interact directly with command-line tools and target machines.

5.20.3 Evaluation

The system is evaluated using a controlled laboratory setup in which a Metasploitable 2 VM is deployed on the same local network as the testing environment. The evaluation focuses primarily on demonstrating functional capability rather than comparative benchmarking against alternative systems. In this scenario, BreachSeek successfully exploited vulnerabilities on the target system and obtained root-level access after executing a sequence of automated actions.

The authors report that the complete attack process required approximately 150,000 tokens of interaction with the language model. The evaluation is qualitative in nature and is intended to illustrate feasibility rather than provide statistically rigorous performance metrics. The study notes that future work will incorporate quantitative evaluation methods using established testing frameworks and certification-style benchmarks such as the OWASP Web Security Testing Guide and the Offensive Security Certified Professional examination.

5.20.4 Strengths

BreachSeek introduces several practical design features relevant to automated PenTesting research:

- *Graph-based multi-agent coordination:* Task distribution across specialised agents helps manage context complexity and supports multi-step workflows.
- *Integrated command execution:* The system can interact directly with PenTesting tools through a shell interface within a controlled environment.
- *Automated reporting:* A Recorder component maintains a summary of actions and generates a final report describing the penetration testing process.
- *Container-based deployment:* Use of Docker and Kali Linux supports reproducible testing environments and flexible system configuration.

5.20.5 Limitations

Several limitations constrain the interpretation of the reported results:

- *Limited evaluation scope:* Experiments are conducted on a single intentionally vulnerable machine rather than a diverse set of targets.
- *Absence of quantitative benchmarking:* The study relies primarily on qualitative demonstration rather than systematic performance metrics.
- *Controlled environment assumptions:* Testing occurs within a local laboratory network without adversarial conditions or defensive monitoring.
- *Dependence on predefined tools:* The system operates by orchestrating existing PenTesting utilities rather than discovering novel exploitation techniques.

5.20.6 Summary

BreachSeek demonstrates a modular multi-agent approach to automated PenTesting in which specialised components coordinate planning, execution, evaluation, and reporting tasks within a containerised testing environment. The system illustrates the feasibility of integrating LLM-driven agents with conventional PenTesting tools to automate multi-step attack workflows. However, the evaluation remains limited to controlled laboratory scenarios, and future work is expected to incorporate quantitative benchmarking and additional evaluation frameworks.

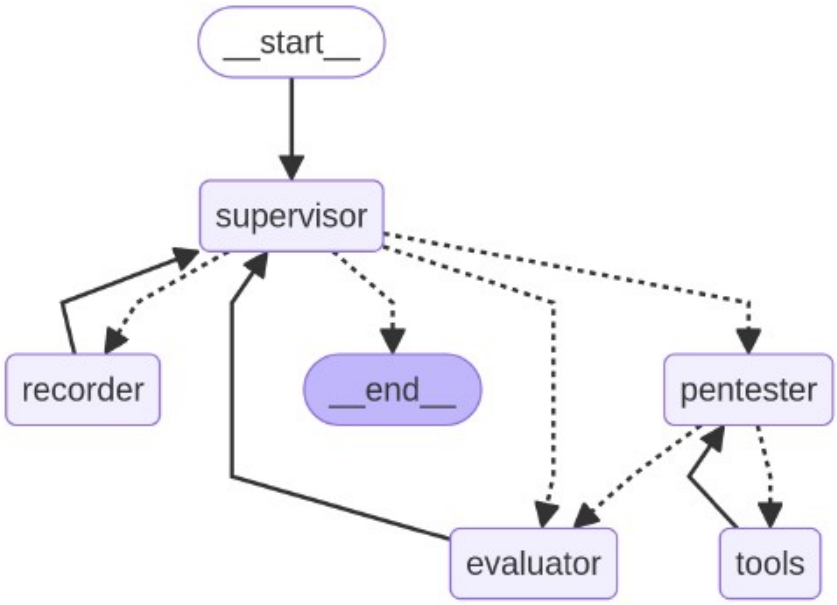


Fig. 22: The workflow of BreachSeek, reproduced from [19] under fair use

5.21 PTGroup

5.21.1 Introduction

PTGroup [167] presents an automated PenTesting framework that integrates LLMs into a multi-agent workflow designed to execute PenTesting tasks through iterative decision cycles. The system is motivated by the operational cost and expertise requirements associated with traditional manual PenTesting, and aims to reduce these burdens through structured automation. The framework adopts

a reasoning-and-action paradigm inspired by the ReAct methodology, in which the system repeatedly performs *Thought–Act–Observe* cycles to guide testing activities and interpret feedback from the target environment.

Rather than introducing novel exploitation techniques, PTGroup focuses on orchestrating existing PenTesting tools within a coordinated workflow. The authors emphasise that the framework leverages the prior knowledge encoded in LLMs to guide decision-making without additional model training. Experimental results demonstrate that the system can adapt to different LLM back-ends and that its performance improves when more capable models are used.

5.21.2 Architecture and Workflow

PTGroup implements a structured multi-agent architecture in which specialised agents correspond to stages of the Thought–Act–Observe loop. The system is designed to run on a Kali Linux environment, allowing agents to interact directly with standard PenTesting utilities through command-line execution (see Fig. 23).

The architecture comprises four principal agents:

- *Master:* Serves as the central decision-making component responsible for generating recommendations and determining subsequent testing actions based on the current system state.
- *Actor:* Converts the Master's recommendations into executable commands and interacts with the target environment through PenTesting tools.
- *Fixer:* Corrects malformed or non-standard command outputs generated by the Actor, using a predefined blacklist to detect known execution errors.
- *Assistant:* Maintains a log of actions and outcomes, recording key information such as IP addresses, open ports, and execution results to support subsequent decision cycles.

The workflow begins when a user provides a target IP address, after which the Master typically initiates reconnaissance activities such as port scanning. The resulting observations are logged by the Assistant and incorporated into the next decision cycle, enabling the system to progress iteratively until a vulnerability is successfully exploited. Fig. 23 illustrates this closed-loop interaction between agents, the Kali Linux environment, and the target system.

To improve robustness across different vulnerability scenarios, the authors introduce a *multiple prompt chains* mechanism. Each prompt chain represents a predefined strategy for handling a class of vulnerabilities, such as password brute-forcing or exploit selection in Metasploit. The system automatically transitions between prompt chains when predefined iteration limits are reached, allowing it to explore alternative strategies when earlier attempts fail (see Fig. 24).

5.21.3 Evaluation

The evaluation is conducted in a controlled virtual laboratory environment in which PTGroup is deployed on a Kali Linux container and tested against intentionally vulnerable systems, including Metasploitable 2, Vulhub environments,

and Windows VMs. The experimental setup uses a single network segment without lateral movement scenarios.

The results demonstrate that PTGroup can successfully exploit several well-known vulnerabilities, including Vsftpd 2.3.4, OpenSSH 4.7, Telnet services, and MS17-010. However, the system fails to exploit certain services, such as Apache 2.2.8, primarily due to incorrect exploit selection or parameter configuration errors. The performance analysis also shows that the success rate increases when more capable language models are used, with GPT-4 producing more reliable outcomes than GPT-3.5 in the same testing scenarios.

The evaluation focuses on demonstrating feasibility and operational behaviour rather than benchmarking performance against alternative systems or testing in adversarial production environments.

5.21.4 Strengths

PTGroup introduces several design features relevant to automated PenTesting research:

- *Structured Thought–Act–Observe workflow:* The iterative decision loop provides a clear mechanism for integrating reasoning, execution, and feedback.
- *Multi-agent role separation:* Distinct agents handle planning, execution, correction, and logging tasks, improving modularity and workflow clarity.
- *Prompt-chain strategy switching:* Predefined prompt sequences allow the system to transition between alternative attack strategies when progress stalls.
- *Compatibility with standard tools:* Deployment on Kali Linux enables direct interaction with widely used PenTesting utilities.

5.21.5 Limitations

Several constraints limit the generality of the reported results:

- *Controlled laboratory evaluation:* Experiments are conducted in isolated virtual environments rather than realistic enterprise networks.
- *Dependence on predefined prompt strategies:* The system relies heavily on manually designed prompt chains, requiring substantial human effort to configure.
- *Limited attack scope:* The evaluation focuses primarily on single-host exploitation scenarios without lateral movement or defensive countermeasures.
- *Sensitivity to model capability:* Performance varies significantly with the underlying LLM, indicating limited robustness across model configurations.

5.21.6 Summary

PTGroup demonstrates a structured multi-agent approach to automated PenTesting in which LLM-driven components coordinate planning, execution, correction, and logging activities within an iterative workflow. The framework illustrates how prompt-based strategy switching can support automated exploitation of common vulnerabilities using existing PenTesting tools. However, the

evaluation remains limited to controlled testing environments, and the system's effectiveness in more complex or adversarial settings is not established.

5.22 ICSSPulse

5.22.1 Introduction

ICSSPulse [148] is a modular, open-source platform designed to support PenTesting activities targeting Industrial Control System (ICS) communication protocols. The system provides a lightweight web-based environment that integrates network discovery, protocol-aware interaction with industrial services, and automated reporting capabilities.

Unlike traditional PenTesting frameworks that primarily target enterprise IT systems, ICSSPulse focuses on Operational Technology (OT) environments where industrial protocols such as Modbus and OPC UA are widely used. These protocols often expose operational data and control interfaces that may be weakly authenticated or improperly segmented within industrial networks. ICSSPulse addresses this gap by enabling controlled and reproducible security assessments of ICS communication channels in simulated environments, thereby facilitating research, training, and educational experimentation in ICS cybersecurity.

A distinctive feature of ICSSPulse is the integration of a LLM (LLM)-assisted reporting component that automatically converts technical findings into structured executive and technical reports with mitigation guidance aligned with the MITRE ATT&CK for ICS framework. This capability demonstrates how LLMs can support the reporting phase of PenTesting workflows by translating low-level technical artefacts into actionable security documentation.

5.22.2 Architecture and Workflow

ICSSPulse is implemented as a modular web-based platform built around a unified graphical user interface (GUI) that coordinates PenTesting activities across multiple backend components. The architecture separates user interaction, protocol processing, and reporting modules, enabling extensibility and reproducibility in experimental settings.

The platform consists of several core modules:

- **Network Scanning Module:** Integrates the RustScan tool within a containerised environment to perform host discovery and port scanning.
- **Modbus Handler:** Implements protocol-aware enumeration and interaction with Modbus services using the `pymodbus` library, supporting read and write operations on coils and registers.
- **OPC UA Handler:** Enables namespace discovery, variable enumeration, and read/write interaction with OPC UA nodes using the `python-opcua` framework.
- **LLM-Assisted Reporting Module:** Aggregates scan and interaction artefacts and generates structured security reports using a GPT-based model.

These components operate through a central orchestration layer that receives user inputs from the GUI and dispatches tasks to protocol-specific handlers. Intermediate results are stored as machine-readable artefacts (e.g., JSON outputs), allowing them to be visualised in the interface or aggregated for later reporting. This modular design allows ICSSPulse to incorporate additional industrial protocols without modifying the user interface or existing modules.

5.22.3 Integration with the PenTesting Lifecycle

ICSSPulse is designed to support multiple stages of the PenTesting lifecycle in ICS environments. The platform provides configurable mechanisms for planning, reconnaissance, enumeration, exploitation, and reporting within a controlled laboratory setting.

- **Planning and Reconnaissance:** Users configure target IP addresses, ports, protocol parameters, and operation types through the web interface.
- **Scanning and Enumeration:** Integrated scanning capabilities identify active hosts and services, while protocol handlers enumerate accessible registers, nodes, and address ranges.
- **Vulnerability Analysis:** Structured outputs enable analysts to identify insecure protocol behaviour, such as unauthenticated Modbus access or exposed OPC UA endpoints.
- **Exploitation:** Protocol handlers support controlled read/write interactions with Modbus registers and OPC UA variables to evaluate potential manipulation of industrial processes.
- **Post-Exploitation Observation:** Device responses and state changes are displayed through the interface, allowing analysts to observe the impact of protocol-level actions.
- **Reporting and Remediation:** Collected artefacts are consolidated into automated reports generated by the LLM-assisted reporting module.

This lifecycle integration allows the platform to simulate realistic ICS PenTesting workflows while maintaining operational safety by operating on simulated industrial services rather than live infrastructure.

5.22.4 Evaluation Methodology

The authors evaluate ICSSPulse using several simulated industrial environments. For Modbus testing, two scenarios are employed: a synthetic Modbus TCP server implemented using the `pymodbus` library and a Factory I/O water-treatment simulation representing a simplified industrial process. These testbeds allow the platform to perform device discovery, register enumeration, and controlled manipulation of process variables.

For OPC UA evaluation, a custom Python-based OPC UA server simulating an industrial production line is deployed. The namespace includes sensor and

actuator nodes representing temperature measurements and motor control variables. Using ICSSPulse, the platform successfully discovers endpoints, enumerates namespace structures, retrieves variable metadata, and performs read/write operations on process nodes.

Finally, the LLM-assisted reporting module aggregates scanning outputs and interaction artefacts to generate both executive summaries and detailed technical reports describing the observed vulnerabilities and recommended mitigation strategies.

5.22.5 Strengths

ICSSPulse contributes several capabilities relevant to LLM-assisted PenTesting in industrial environments:

- *ICS-Specific Focus:* Targets industrial communication protocols such as Modbus and OPC UA rather than traditional enterprise services.
- *Modular Architecture:* Separates scanning, protocol interaction, and reporting components to support extensibility.
- *Web-Based Accessibility:* Provides a user-friendly interface that facilitates training and experimentation.
- *LLM-Assisted Reporting:* Demonstrates how language models can automate the translation of technical findings into structured security reports.

5.22.6 Limitations

The authors note several limitations in the current version of the platform:

- *Protocol Scope:* The system currently supports only Modbus and OPC UA protocols.
- *Simulation-Based Evaluation:* Experiments rely on simulated industrial environments rather than large-scale operational deployments.
- *Limited Post-Exploitation Coverage:* System-level attacks such as PrivEsc or lateral movement are not implemented.
- *Scalability Constraints:* The Python-based implementation and containerised execution environment may limit performance in larger infrastructures.

5.22.7 Summary

ICSSPulse demonstrates how modular PenTesting platforms can be adapted for ICS environments and augmented with LLM capabilities to automate reporting and analysis tasks. By combining protocol-aware enumeration, controlled exploitation of industrial communication channels, and automated report generation, the platform provides a practical environment for studying ICS security assessment workflows. While currently limited to simulated environments and a small set of protocols, ICSSPulse illustrates the potential of integrating LLM-assisted components into industrial PenTesting pipelines.

5.23 AutoPenBench

5.23.1 Introduction

AutoPenBench [54] is an open benchmark framework designed to evaluate generative agents for automated PenTesting in a standardised and reproducible manner. While prior work has demonstrated LLM-driven PenTesting capabilities, meaningful comparison has been hindered by the absence of common evaluation environments and metrics. AutoPenBench addresses this gap by providing a collection of realistic vulnerable systems together with a structured methodology for measuring agent progress and success.

The benchmark comprises 33 tasks implemented as containerised vulnerable systems, spanning both simplified 'in-vitro' scenarios and realistic real-world vulnerabilities derived from publicly disclosed CVEs. Each task follows a CTF paradigm in which the agent must discover, exploit, and retrieve a hidden flag. The framework also supports evaluation of different agent architectures and LLM back-ends under identical conditions, enabling controlled comparison of autonomy, reliability, and performance.

5.23.2 Architecture and Workflow

AutoPenBench defines a complete virtual PenTesting infrastructure consisting of (i) an agent workstation running Kali Linux and standard offensive tools, (ii) one or more vulnerable containers representing target systems, and (iii) an isolated virtual network enabling unrestricted interaction across protocols. Agents can execute arbitrary commands within this environment rather than being limited to predefined tools, allowing more realistic attack behaviour.

A key contribution of the framework is its milestone-based evaluation methodology. Each task is decomposed into:

- *Command Milestones*, representing specific actions (e.g., discovering a host, exploiting a vulnerability);
- *Stage Milestones*, representing higher-level phases of the attack lifecycle (e.g., discovery, infiltration, exploitation, flag capture).

Agent progress is quantified using a Progress Rate metric based on achieved milestones, enabling fine-grained analysis even when tasks are not fully solved. The framework also supports automatic verification of milestone completion using LLM-as-a-Judge evaluation with optional manual correction.

To demonstrate the benchmark's utility, the authors implement two generative agent architectures based on a ReAct-style reasoning loop: a fully autonomous agent and a semi-autonomous agent that accepts human guidance through sub-task instructions.

5.23.3 Evaluation

Experiments are conducted across all 33 tasks using GPT-4o as the underlying model. Results reveal a substantial gap between autonomous and human-assisted performance. The fully autonomous agent achieves an overall success rate of approximately 21%, performing moderately on simple in-vitro tasks but solving only a single real-world vulnerability. In contrast, the assisted agent attains a 64% success rate, with particularly strong results on realistic CVE-based scenarios.

Detailed analysis shows that both agents perform reliably in early reconnaissance stages (e.g., host discovery) but frequently fail during vulnerability exploitation and parameter configuration. The benchmark also exposes variability across repeated runs, highlighting the impact of LLM stochasticity on operational reliability.

5.23.4 Strengths

AutoPenBench offers several contributions for research on LLM-based offensive security systems:

- *Standardised Evaluation Framework:* Provides a common platform for comparing PenTesting agents under controlled conditions.
- *Realistic Test Environment:* Uses containerised vulnerable systems and unrestricted command execution to approximate real PenTesting workflows.
- *Fine-Grained Metrics:* Milestone-based scoring captures partial progress, enabling deeper diagnostic analysis than binary success measures.
- *Support for Human-AI Collaboration Studies:* Enables direct comparison between autonomous and assisted agent paradigms.

5.23.5 Limitations

The authors acknowledge several limitations of the benchmark:

- *Limited Scenario Coverage:* Although diverse, the task set cannot fully represent the complexity of large enterprise environments.
- *Dependence on Known Vulnerabilities:* Real-world tasks rely on publicly documented CVEs with available exploits, potentially favouring memorisation.
- *Evaluation Overhead:* Running large benchmark campaigns requires numerous LLM calls and extensive experiment execution.
- *Residual Human Oversight:* Manual intervention may still be required for evaluation correction and assisted-agent operation.

5.23.6 Summary

AutoPenBench provides one of the first comprehensive open benchmarks for evaluating LLM-driven PenTesting agents. Its containerised infrastructure, milestone-based metrics, and support for both autonomous and human-assisted modes

enable rigorous comparison of agent capabilities while revealing current limitations in reliability and exploitation performance. As such, the framework serves as an important foundation for future research on trustworthy and reproducible autonomous offensive security systems.

5.24 CyberExplorer

5.24.1 Introduction

CyberExplorer [126] is a behaviour-centric benchmark and evaluation framework designed to assess the offensive security capabilities of LLM-based agents in realistic, open-ended attack environments. Unlike prior benchmarks that present isolated challenges with predefined goals, CyberExplorer simulates a multi-target system in which agents must autonomously discover vulnerable services, prioritise targets, and execute exploitation without prior knowledge of vulnerability locations.

The framework addresses a key limitation in existing evaluations: most prior studies assess agents in closed-world settings where success is measured solely by flag retrieval from a single service instance. In contrast, CyberExplorer introduces an *Open Environment Offensive Security Task* in which multiple vulnerable services coexist within a shared environment, requiring reconnaissance, hypothesis revision, and decision-making under uncertainty. This approach aims to better approximate real-world PenTesting scenarios.

5.24.2 Architecture and Workflow

CyberExplorer comprises two tightly coupled components: a realistic simulation environment and a reactive multi-agent system. The environment is implemented as a VM hosting 40 web-based vulnerable services deployed as independent Docker containers and exposed via network ports, forming a partially observable attack surface. Agents must infer service identities and vulnerabilities through probing and interaction feedback rather than explicit guidance.

The agent architecture follows a reconnaissance–analysis–execution workflow coordinated by supervisory components. As illustrated in Fig. 27, the system includes specialised roles such as reconnaissance agents, executor agents, a supervisor, and a critic that provides trajectory corrections. During reconnaissance, entry points (e.g., open ports and services) are identified and queued for exploration. Exploration is then performed in parallel by sandboxed agent teams interacting with individual services.

Each team consists of a sequence of short-lived agents that inherit knowledge from predecessors, including discovered vulnerabilities, failed attempts, and contextual findings. To mitigate inefficient search, the framework introduces mechanisms such as:

- *Supervisor-guided task directives* derived from prior exploration history;
- *Self-reflection checkpoints* triggered at predefined budget thresholds;

– *Critic interventions* that suggest alternative strategies when progress stalls;
– *Early termination heuristics* to abandon unproductive targets.

Agents operate within containerised environments equipped with offensive security tools, enabling realistic interaction with target services. This design supports both parallel exploration across entry points and sequential refinement within each target.

5.24.3 Benchmark Design

The CyberExplorer benchmark is constructed from 40 web-based CTF challenges drawn from multiple sources, including NYU CTF Bench, Google CTF, Hack The Box, HKCERT CTF, Project Sekai CTF, and CodeGate. The challenges span common vulnerability classes such as injection flaws, authentication bypasses, logic errors, and misconfigurations.

Unlike conventional benchmarks, success depends not only on exploitation ability but also on reconnaissance effectiveness, target selection, and resilience to false positives. Because multiple vulnerable services coexist in a single environment, agents must prioritise among competing hypotheses rather than solving independent tasks sequentially.

5.24.4 Evaluation Methodology

CyberExplorer evaluates agents across a broad set of behavioural metrics rather than binary success alone. Experiments are conducted under fixed budgets to ensure fair comparison across models. Performance measures include correctness (true/false positives), interaction efficiency, coordination behaviour, failure persistence, and vulnerability discovery signals.

Empirical results reveal substantial differences across models. For example, higher-performing models demonstrate rapid convergence and stable reasoning with fewer interaction rounds, whereas weaker models exhibit prolonged exploration with low precision. Time-to-first-flag and agent utilisation metrics further distinguish early efficiency from sustained reasoning quality. Dead-end trajectories typically consume significantly more rounds and cost than successful ones, indicating persistence in incorrect hypotheses.

Importantly, the framework also measures partial intelligence extraction. Agents often surface misconfigurations or vulnerability indicators even when exploitation fails, providing useful reconnaissance output analogous to real Pen-Testing reports.

5.24.5 Strengths

CyberExplorer contributes several capabilities for evaluating LLM-based offensive systems:

– *Open-Environment Evaluation:* Simulates realistic multi-target systems rather than isolated tasks.

- *Reactive Multi-Agent Coordination:* Supports parallel exploration with supervisor and critic guidance.
- *Behaviour-Centric Metrics:* Analyses reasoning efficiency, coordination dynamics, and failure modes.
- *Security-Relevant Signal Extraction:* Captures vulnerability findings even without successful exploitation.

5.24.6 Limitations

The authors acknowledge several constraints:

- *Web-Centric Scope:* The benchmark focuses on web services and does not cover client-side, kernel, or post-exploitation scenarios.
- *Budget Sensitivity:* Fixed interaction limits may influence observed behaviour and persistence patterns.
- *Trace-Based Analysis:* Evaluation relies on observable actions rather than internal model reasoning.
- *Controlled Environment:* Results may not fully generalise to real-world systems with dynamic conditions.

5.24.7 Summary

CyberExplorer introduces a behaviour-focused paradigm for evaluating autonomous offensive security agents in realistic, open-ended environments. By combining a multi-service benchmark with a reactive multi-agent framework, it exposes aspects of agent performance—such as coordination quality, hypothesis management, and persistence under uncertainty—that are not visible in traditional success-only benchmarks. The framework thus provides a foundation for more comprehensive assessment of LLM-driven PenTesting systems and highlights the importance of analysing how agents operate, not merely whether they succeed.

5.25 LLM Pentest Benchmark

5.25.1 Introduction

Isozaki et al. [75] present one of the first open benchmarks aimed at evaluating LLMs for end-to-end PenTesting under constrained human assistance. Prior work typically examined isolated tasks—such as exploitation or PrivEsc—or relied heavily on human operators, making it difficult to measure true autonomy (see Fig. 29). The authors argue that progress in LLM-based offensive security requires a standardised, reproducible benchmark covering the major technical phases of host-based PenTesting, especially reconnaissance, exploitation, and PrivEsc.

To address this gap, the study introduces a publicly available benchmark based on vulnerable VMs and evaluates it using the PentestGPT framework with two state-of-the-art models: GPT-4o and Llama 3.1-405B. The results highlight both the promise and current limitations of LLMs, showing that neither model can complete an end-to-end penetration test without substantial human assistance.

5.25.2 Benchmark Design and Methodology

The benchmark is constructed using freely available VulnHub machines to ensure reproducibility and accessibility, avoiding paywalled platforms. Tasks are derived from public walkthroughs and categorised into four major PenTesting phases: reconnaissance, general techniques, exploitation, and PrivEsc. The distribution of tasks reflects real-world workflows, with reconnaissance dominating early stages and exploitation and PrivEsc appearing later.

Strict rules are defined to minimise human involvement while maintaining feasibility. For example, the number of attempts per task is limited, GUI-based tools may require human interaction, and full outputs—such as HTML content—must be provided to the model when visiting websites. Unlike earlier benchmarks that halted evaluation after failure, this framework assesses performance across all tasks to obtain a comprehensive profile of strengths and weaknesses.

The benchmark adopts PentestGPT's task structure but clarifies operational boundaries, defines success criteria, and standardises evaluation procedures. This approach enables systematic comparison across models and PenTesting stages.

5.25.3 Evaluation

Experiments compare GPT-4o and Llama 3.1-405B across machines of varying difficulty. Overall, Llama 3.1-405B demonstrates, according to this study, superior performance, particularly on easy and medium scenarios, although both models degrade sharply as complexity increases. Success rates are highest for reconnaissance and general techniques, while exploitation and PrivEsc remain challenging.

Notably, neither model achieves root privileges on any machine in a fully autonomous manner, underscoring the gap between current capabilities and real-world PenTesting. Performance analysis shows that models often become less effective as tasks progress, suggesting difficulties with long-horizon reasoning and state tracking.

5.25.4 Ablation Studies

The authors conduct ablation studies (see Fig. 28) to identify factors limiting performance and to improve the PentestGPT framework. Three cumulative enhancements are evaluated:

- *Summary Injection:* Maintaining a persistent summary of prior progress to mitigate forgetting.
- *Structured Task Lists:* Replacing natural-language task trees with explicit to-do lists to reduce hallucinations and improve planning.
- *Retrieval-Augmented Context (RAG):* Incorporating external knowledge from security resources such as HackTricks to support decision-making.

Results indicate that the cumulative ablation 3 setting, which adds RAG on top of summary injection and structured task management, produces the strongest overall performance on the evaluated boxes, although the authors remain cautious about whether structured task lists are always beneficial.

5.25.5 Strengths

The benchmark contributes several features:

- *Broad Workflow Coverage:* Covers major technical stages of PenTesting—from reconnaissance through exploitation and privilege escalation—rather than focusing on a single isolated phase.
- *Reproducibility:* Uses publicly available vulnerable machines and open-source resources.
- *Detailed Performance Analysis:* Provides fine-grained insights across task categories and difficulty levels.
- *Architectural Insights:* Demonstrates the importance of memory mechanisms, structured planning, and external knowledge.

5.25.6 Limitations

Several constraints limit the current study:

- *HITL Requirement:* Some tasks still require human execution or interpretation.
- *Static Vulnerability Scenarios:* Evaluation relies on known vulnerable systems rather than live environments.
- *Limited Autonomy:* Models cannot complete end-to-end tests independently.
- *Potential Data Leakage:* Public walkthroughs may overlap with training data.
- *Limited Ablation Scope:* Due to cost and time constraints, the ablation study is conducted on only two boxes.

5.25.7 Summary

This work establishes a foundational benchmark for evaluating LLM-driven PenTesting systems and demonstrates that current models remain far from autonomous offensive capability. While LLMs show competence in early-stage reconnaissance, they struggle with exploitation, PrivEsc, and long-horizon coordination. The study highlights the critical role of structured planning, persistent memory, and retrieval mechanisms, providing a roadmap for future research toward reliable automated PenTesting.

5.26 Cybench: Evaluating Cybersecurity Capabilities and Risks of LLMs

5.26.1 Introduction

Zhang et al. [179] introduce *Cybench*, an open benchmarking framework designed to systematically evaluate the cybersecurity capabilities and risks of large language models (LLMs). Rather than focusing on a single exploitation scenario, Cybench evaluates LLM-based agents across a diverse collection of professional-level CTF challenges representing realistic security tasks. The benchmark aims to provide a reproducible and standardised evaluation environment for measuring how effectively LLMs can perform complex cybersecurity activities such as vulnerability analysis, exploitation, reverse engineering, and digital forensics. All benchmark artefacts—including task definitions, evaluation scripts, and supporting infrastructure—are publicly released to support transparent and repeatable experimentation[89].

5.26.2 Architecture and Workflow

Cybench is implemented as an automated evaluation pipeline that integrates LLM agents with controlled execution environments. Each benchmark task is defined by three primary components: a textual task description specifying the objective (typically capturing a flag), a set of starter files available to the agent, and an evaluator that verifies task completion. Starter files may include local artefacts (e.g., source code or configuration files) as well as references to remote task servers hosting vulnerable services (see Fig. 30).

Tasks are instantiated in a sandboxed environment built around a Kali Linux Docker container, which serves as the agent's execution environment. The container provides standard command-line tools and allows the agent to interact with both local files and remote task servers through network requests. Remote services are hosted in separate Docker containers connected within the same virtual network.

The evaluation agent follows an iterative *act–execute–update* loop. At each step, the agent generates an action (typically a shell command) based on its current memory, which includes the initial task prompt and recent interaction history. The command is executed within the environment, producing an observation that is returned to the agent and incorporated into its memory for subsequent reasoning steps.

Task success is determined automatically through evaluators that verify whether the agent submits the correct solution or produces a unique flag string indicating successful exploitation. In addition to binary task success, the framework also records metrics such as token usage and execution time to support comparative evaluation. The framework is model-agnostic and supports both proprietary and open-source LLMs accessed through APIs or local deployments.

[89] `https://cybench.github.io/`

5.26.3 Evaluations and Results

Using Cybench, the authors evaluate 8 modern LLMs on a benchmark comprising 40 professional-level CTF tasks. These tasks span several common cybersecurity domains, including web exploitation, cryptography, reverse engineering, digital forensics, and binary exploitation. The evaluation includes both proprietary models (e.g., GPT-4o, Claude 3 family, Gemini models) and open-source models (e.g., LLaMA-based and Mixtral models).

Results indicate that while recent LLMs demonstrate emerging capabilities in solving structured security challenges, their performance remains limited. Even the strongest models solve only a subset of the tasks, highlighting the difficulty of multi-step reasoning and tool interaction required in realistic cybersecurity problems. Proprietary models generally outperform open-source alternatives, although significant variability is observed across task categories.

5.26.4 Strengths

Cybench introduces several valuable contributions to the study of LLM capabilities in cybersecurity:

- *Comprehensive benchmark design.* By using professional-level CTF challenges, the benchmark captures a diverse range of practical cybersecurity tasks.
- *Reproducible evaluation.* All benchmark artefacts, including tasks, evaluation scripts, and infrastructure, are publicly released to enable transparent comparison between models.
- *Realistic execution environment.* Sandboxed environments allow LLM agents to interact with tools and systems in a controlled yet realistic setting.
- *Model-agnostic framework.* The evaluation pipeline supports both proprietary and open-source LLMs, facilitating broad comparative studies.

5.26.5 Limitations

Despite its contributions, several limitations remain:

- *Benchmark scope.* The evaluation is based on CTF-style tasks, which may not fully capture the complexity of real-world PenTesting or large-scale intrusion campaigns.
- *Limited multi-stage attack evaluation.* While individual tasks may involve multiple steps, the framework does not yet evaluate long, multi-phase attack chains typical of real engagements.
- *Tool interaction constraints.* Agents operate within predefined execution environments and tool sets, which may differ from the flexibility available to human security professionals.
- *Performance variability.* Results vary significantly across task categories, indicating that current LLMs remain inconsistent in complex cybersecurity reasoning tasks.

5.26.6 Summary

Overall, Cybench provides an open and reproducible benchmark for evaluating LLM capabilities in cybersecurity contexts. The framework enables systematic comparison of models on realistic security challenges and offers a foundation for future research into LLM-assisted offensive and defensive security workflows. The results highlight both the emerging potential of LLM-based agents and the significant limitations that remain in their ability to solve complex cybersecurity tasks autonomously.

5.27 HackSynth

5.27.1 Introduction

HackSynth [100] is an LLM-driven autonomous PenTesting agent evaluated using controlled CTF-style benchmarks. Rather than targeting real enterprise environments, the system focuses on reproducible experimentation across structured tasks, aiming to quantify (i) how effectively different base models can solve challenges end-to-end, and (ii) how inference-time parameters (e.g., temperature and top-p) affect success, diversity of commands, and operational safety.

5.27.2 Architecture and Workflow

HackSynth follows a lightweight two-module control loop. A *Planner* proposes the next command to execute (formatted in explicit command tags), while a *Summariser* condenses terminal feedback into a compact state for the next planning step (see Fig. 31). The agent operates inside a restricted, non-interactive shell, which intentionally prevents typical human workflows (e.g., using interactive editors). As a result, HackSynth often substitutes CLI-native alternatives (e.g., auto-formatters or stream editors) when fixing code or patching files.

For benchmark runs, the agent is allowed a fixed budget of iterative plan–execute–summarise steps (20 loops). The study also explores how observation-window limits interact with summarisation, and how sampling choices influence rare-command usage and error rates.

5.27.3 Evaluation

HackSynth is evaluated on two CTF-derived benchmarks: (i) a PicoCTF subset (120 tasks) spanning categories such as general, forensics, crypto, web, reversing, and binary exploitation; and (ii) an OverTheWire benchmark (80 tasks) that emphasises staged interaction (often requiring frequent use of commands such as `ssh` and `curl`). The authors benchmark eight instruction-tuned models (including GPT-4o, GPT-4o-mini, and several open-weight families).

With the benchmark settings fixed to temperature = 1 and top-p = 0.9, GPT-4o achieves the best overall performance, solving 41/120 PicoCTF challenges and 32/80 OverTheWire challenges. Larger open models (e.g., Llama-3.1-70B

and Qwen2-72B) trail GPT-4o but outperform smaller local models on average. The paper also reports that (a) performance is broadly stable up to temperature 1 but degrades at higher temperatures as command quality worsens and failure rates rise, and (b) increasing top-p slightly improves completion and increases the likelihood of selecting rarer commands.

5.27.4 Strengths

HackSynth offers several contributions for the evaluation of LLM-aided offensive-security agents:

- *Reproducible agent benchmark:* A clear, repeatable setup for comparing multiple LLM backbones on the same task suites.
- *Explicit control loop:* A simple Planner–Summariser design that isolates key behaviours (command generation, feedback compression, and iterative refinement).
- *Parameter sensitivity analysis:* Systematic exploration of temperature, top-p, and observation-window sizing, linking them to success, errors, and command diversity.
- *Behavioural insights:* Examples showing how constraints (non-interactive shell) force tool substitutions and can yield surprisingly creative command-line strategies.

5.27.5 Limitations

The authors highlight several constraints that affect interpretation and deployment:

- *External validity:* CTF tasks may not reflect real penetration tests (e.g., multi-host uncertainty, stealth, and socio-technical constraints), and some solutions may benefit from memorisation.
- *Safety and operational risk:* At higher temperatures, the Planner can generate destructive or environment-breaking commands (e.g., deleting/moving binaries or altering configurations), occasionally rendering the environment unusable.
- *Model-dependent behaviours:* Some models exhibit risky tendencies (e.g., frequent `sudo`) or may refuse to output commands for ethical-policy reasons, affecting measured performance.
- *Interaction constraints:* The non-interactive shell simplifies control and containment but also limits realism (e.g., no interactive editors), shaping the solution strategies.

5.27.6 Summary

HackSynth demonstrates that a minimal Planner–Summariser agent can solve a non-trivial fraction of CTF challenges using only iterative command-line interaction, while exposing sharp trade-offs between performance, exploration, and

safety. Its key value for LLM-powered PenTesting surveys is methodological: it provides an empirical, parameter-sensitive benchmark of agentic command execution across multiple foundation models, while underscoring that robustness and containment become harder as sampling-induced variability increases.

6 Classification of LLM-aided PenTesting Systems

6.1 Introduction

This section surveys 27 representative LLM-aided PenTesting systems, including peer-reviewed papers (see Table 7) and recent preprints (see Table 8), capturing the current state of AI-driven offensive security research.

6.2 PenTesting Systems vs. Benchmarking and Evaluation Frameworks

A conceptual distinction is necessary between *LLM-aided PenTesting systems* and *LLM-aided benchmarking or evaluation frameworks*. Although both appear in the recent literature on AI-driven offensive security, their objectives and operational roles differ substantially.

LLM-aided PenTesting systems aim to perform or assist real attack activities within a PenTesting workflow. These systems typically integrate LLM reasoning with external tools, exploit modules, or command execution environments in order to discover vulnerabilities, obtain initial access, escalate privileges, or interact with target applications. Examples include systems targeting specific phases such as foothold acquisition (e.g., PenTest++ or RapidPen), post-exploitation (e.g., Wintermute or PenTest2.0), or web-application attacks (e.g., PenForge or AutoPT). In such systems, the LLM participates directly in the attack process by generating commands, planning actions, interpreting results, or orchestrating tools.

By contrast, **LLM-aided benchmarking and evaluation frameworks** are designed primarily to measure the capabilities, limitations, and risks of LLMs in offensive security contexts. Rather than carrying out real attacks against external targets, these platforms provide controlled environments, curated tasks, or simulated scenarios in which models can be systematically evaluated. Systems such as Cybench, AutoPenBench, and the LLM Pentest Benchmark fall into this category. Their primary objective is reproducible experimentation and comparative evaluation across models or agent architectures.

In practice, the boundary between the two categories is not always strict. Some benchmarking platforms include prototype agents capable of performing simplified attacks within sandboxed environments, while some PenTesting systems incorporate evaluation components to measure success rates or reasoning quality. Nevertheless, distinguishing between operational PenTesting systems and benchmarking frameworks clarifies the different research goals represented in the literature.

For completeness, this survey includes both categories. Operational PenTesting systems constitute the majority of the taxonomy presented in Table 9, while benchmarking frameworks are discussed separately as a distinct class because of their role in evaluating LLM offensive capabilities rather than deploying them in real PenTesting workflows.

6.3 Taxonomy

As shown in Table 9, we categorise the surveyed systems into seven groups according to their primary operational scope and capability[90].

1. **Early LLM-as-Advisor Systems** Early LLM-aided PenTesting research treated LLMs as advisory assistants rather than autonomous agents. In these systems, the LLM provides guidance, explanations, and suggested actions, while human testers perform decision-making and execution. This represents the initial stage of AI-driven offensive security, demonstrating feasibility without autonomy. *GenAI for PenTesting* [70] presents a conceptual framework outlining how LLMs can support vulnerability analysis, attack planning, and reporting, establishing a foundation for later systems. *AI-Augmented Ethical Hacking* [12] demonstrates practical GenAI assistance across multiple PenTesting phases under human supervision, including reconnaissance and exploitation guidance, highlighting productivity gains alongside the need for expert oversight. *PentestGPT* [40] provides interactive guidance across the PenTesting workflow, acting as a copilot that suggests strategies and interprets results while execution remains human-driven.
2. **Initial Exploitation Systems** This group comprises LLM-driven systems targeting the initial compromise phase, aiming to obtain a foothold in externally reachable targets through vulnerability exploitation, credential harvesting, or automated attack orchestration. *PenTest++* [6, 7] performs end-to-end external reconnaissance and exploitation, guiding the transition from service discovery to system access. *RapidPen* [103] emphasises "IP-to-shell" automation, converting reachable hosts into interactive access with minimal human intervention. *AutoAttacker* [171] orchestrates LLM-guided attack sequences against exposed services, focusing on vulnerability exploitation to establish entry. *VulnBot* [83] automates vulnerability discovery and exploitation workflows to obtain initial access, demonstrating the feasibility of LLM-driven intrusion pipelines. Collectively, these systems prioritise foothold acquisition over lateral movement or post-exploitation, representing early attempts at autonomous entry. *BreachSeek* [19] employs a multi-agent architecture to execute PenTesting commands against vulnerable targets in

[90] We observe that many of these systems have been proposed and developed largely in parallel; indeed, even arXiv submissions do not constitute a reliable indicator of priority, as some authors choose not to release preprints. Moreover, certain arXiv postings appear intended primarily to establish precedence, even though the corresponding work may still be ongoing or not yet realised

a controlled environment, demonstrating automated vulnerability exploitation and foothold establishment through coordinated agent workflows. *PT-Group* [167] introduces a multi-agent, prompt-chain-driven framework that automates vulnerability exploitation against a user-specified target host, using iterative Thought–Act–Observe cycles to execute scanning and exploitation steps until system compromise is achieved within a controlled laboratory environment.

3. **Post-Exploitation Systems** Such schemes operate after a foothold is obtained, focusing on PrivEsc, credential abuse, lateral movement, and internal dominance within compromised environments. *Wintermute* [59] represents an early autonomous PrivEsc prototype that assumes an existing low-privilege account and attempts to elevate privileges via a closed LLM–SSH loop. The agent generates shell commands, observes execution outputs, and iteratively refines its strategy to discover misconfigurations and spawn a root shell. *PenTest2.0* [11] is a post-exploitation framework that automates multi-turn Linux PrivEsc through iterative reasoning and command execution under operator governance. Starting from a low-privilege shell, the system repeatedly proposes actions, executes them, and adapts based on feedback, aiming to achieve root access while maintaining transparency and control. *HackingBuddyGPT* [65] provides a dedicated benchmark and prototype for autonomous Linux PrivEsc. Operating from a compromised user context, the system evaluates whether LLMs can independently identify and exploit local vulnerabilities to obtain admin privileges in controlled environments. *Cochise* [62] extends the post-exploitation paradigm to enterprise-scale networks under an Assumed Breach model. Rather than targeting perimeter intrusion, it focuses on internal exploration, credential abuse, lateral movement, and domain dominance within Active Directory environments using a planner–executor architecture. Collectively, these systems focus on post-compromise host and internal-network operations, showing that LLMs can chain actions for PrivEsc but remain constrained by reliability and deployment limits.
4. **Web-Focused Systems** This category targets web applications through black-box HTTP interaction, automated vulnerability discovery, and exploitation of OWASP-style flaws. *PenForge* [71] performs autonomous web PenTesting by dynamically instantiating specialised agents based on reconnaissance results. Operating against real-world CVEs, it interacts with applications via HTTP requests and browser automation to identify and exploit vulnerabilities. *AutoPT* [166] provides a fully automated framework for web vulnerability assessment, combining LLM reasoning with tool orchestration to discover and exploit weaknesses in target applications without manual intervention. *PenHeal* [72] focuses on automated vulnerability discovery and remediation workflows, including web application flaws, demonstrating end-to-end analysis from detection to exploitation guidance. *LLM Agents Hack Websites* [48] demonstrates autonomous browser-based exploitation of web targets, identifying issues such as injection and access-control flaws. *LLM Agents Exploit 1-day Vulns* [47] evaluates exploitation of

recently disclosed web vulnerabilities, showing feasibility against unpatched flaws. *HPTSA* [186] extends this via multi-agent collaboration to discover and exploit previously unknown web vulnerabilities.

5. **Benchmarking and Evaluation Frameworks** This group includes platforms designed to assess the capabilities, limitations, and risks of LLM-driven PenTesting systems rather than performing attacks directly. Such frameworks provide standardised tasks, datasets, and environments for reproducible evaluation and cross-system comparison. *Cybench* [179] offers a comprehensive benchmarking suite covering offensive and defensive cybersecurity tasks, enabling systematic assessment of LLM performance across vulnerability analysis, exploitation reasoning, and incident-response scenarios. *LLM Pentest Benchmark* [75] focuses specifically on PenTesting workflows, providing structured challenges to measure how effectively LLM-based systems can plan and execute attack steps under controlled conditions. *AutoPenBench* [54] introduces an evaluation framework for automated PenTesting agents, emphasising reproducibility and objective comparison across different architectures and model configurations. *CyberExplorer* [126] provides a simulated attack environment for benchmarking LLM offensive capabilities in realistic scenarios, assessing performance, robustness, and potential security risks.
6. **Emerging and Specialised Systems** These systems address environments beyond conventional enterprise IT and web applications, including wireless networks and other domain-specific infrastructures. *WiFiPenTester* [10] focuses on automated wireless-network PenTesting, orchestrating attacks against Wi-Fi authentication protocols and configurations. Operating in a physical-proximity threat model, the system integrates reconnaissance, handshake capture, and password-recovery workflows using domain-specific tools, demonstrating the applicability of LLM-driven agents to RF-based environments. *ICSSPulse* [148] extends LLM-assisted PenTesting to industrial control systems (ICS) and operational technology (OT) environments, providing a modular platform that combines network scanning with protocol-aware analysis of industrial services such as Modbus and OPC UA for asset discovery and controlled interaction. It also employs LLM-based components to interpret findings and generate structured technical reports with mitigation guidance, demonstrating the applicability of LLM agents to cyber-physical infrastructures. Unlike network- or host-centric systems, specialised agents must handle heterogeneous data, safety constraints, and domain-specific paths; together, *WiFiPenTester* and *ICSSPulse* signal a shift toward RF and cyber-physical targets (ICS/OT, IoT, CPS).
7. **Attempts at End-to-End (Near-Autonomous) Systems** A few systems attempt to coordinate multiple PenTesting phases within a single framework, but none achieves full autonomy in realistic settings. *PenHeal* [72] employs a planner–executor loop to discover and exploit vulnerabilities from a target IP with minimal input. However, evaluation is limited to controlled lab environments, and remediation actions are not executed automatically, leaving true end-to-end autonomy unrealised. *PenTest++* [6,7] uses a mixed-

initiative approach combining conventional tools with LLM guidance across major phases, requiring human approval for critical actions and omitting deeper post-exploitation capabilities. *Incalmo* [140] introduces a multi-host red teaming framework that separates high-level planning from execution by delegating tasks to specialised agents. The system demonstrates strong performance in controlled benchmark environments, successfully acquiring at least one critical asset in 37 of 40 generated enterprise-like networks. Nevertheless, the architecture relies on predefined task abstractions, curated environments, and supporting services, and therefore does not yet achieve fully autonomous end-to-end PenTesting in realistic operational settings.

7 Cross-Comparison of LLM-aided PenTesting Systems

7.1 Attack-Phase Emphasis

Table 10 categorises LLM-aided PenTesting systems by the primary phase of the offensive lifecycle they target, including initial exploitation, post-exploitation, mixed end-to-end workflows, and benchmarking platforms. The table shows that many systems concentrate on early attack stages—particularly vulnerability exploitation—while a smaller subset addresses post-exploitation activities such as PrivEsc or enterprise assumed-breach scenarios, and relatively few support comprehensive multi-phase operations. Overall, current research emphasises achieving initial access rather than sustaining long-horizon campaigns across the full PenTesting lifecycle.

7.2 Capability Coverage

Table 11 compares core capabilities of LLM-aided PenTesting systems across major attack phases, showing that most approaches emphasise early stages such as scanning and exploitation, while relatively few provide comprehensive end-to-end support including post-exploitation and reporting. Entries for Cybench, the LLM Pentest Benchmark, and CyberExplorer are marked **N/A** because these are benchmarking frameworks rather than operational PenTesting systems: they define tasks and environments for evaluating agents but do not themselves execute the attack lifecycle, making phases such as exploitation or persistence inapplicable rather than absent. AI-Augmented Ethical Hacking and GenAI for PenTesting are likewise labelled **N/A** as they present conceptual models or PoC studies rather than standalone automated tools. The enumeration column was omitted due to space constraints, since target enumeration is a fundamental capability supported by virtually all operational systems.

7.3 Autonomy, Architecture and Governance

Table 12 compares LLM-aided PenTesting systems by autonomy level, architectural design, execution model, and governance controls. It reveals a predominance of semi-autonomous and HITL designs based on planner–executor or

multi-agent architectures and tool-mediated execution, with varying degrees of safety oversight, while fully autonomous systems with strong governance remain comparatively rare.

7.4 Reasoning and Guidance

Table 13 compares the use of state-of-the-art reasoning and guidance techniques across LLM-aided PenTesting systems, including CoT, ReAct, planner–executor designs, human hints, RAG, tool invocation, and task-tree tracking. It shows that most systems rely primarily on prompt-level reasoning—especially CoT—often combined with ReAct-style interaction and programmatic tool use, while more advanced guidance mechanisms such as RAG, HITL hints, and explicit task-tree planning appear only in a minority of systems. This pattern favours short-horizon reasoning, with long-horizon planning and external memory still limited.

7.5 Platform, Target, Openness, and Evaluation

Table 14 provides a consolidated comparison of representative LLM-driven PenTesting systems (published and preprint) across execution platform, target surface, prototype availability, and evaluation methodology. The results show a clear shift from early Linux-centric prototypes toward cross-platform designs addressing diverse domains, including host, web, enterprise, wireless, and benchmarking scenarios, indicating an increasing emphasis on generality and real-world applicability. Despite this broad coverage, only a number of systems release fully open-source implementations, with many remaining closed or only partially disclosed. Evaluations are predominantly conducted in controlled laboratory environments, benchmarks, or simulated settings rather than uncontrolled operational deployments. Overall, research prioritises safety and controlled breadth over real-world validation and reproducibility, limiting transparency despite rapid progress.

8 Discussion and Challenges

Despite recent successes, today's LLMs still exhibit fundamental limitations that constrain their usefulness for autonomous PenTesting. Many of the issues are inherent to the underlying architecture and training procedures, while others derive from the way models are deployed and accessed. This section brings together observations from recent PenTesting studies and broader AI research to outline key shortcomings.

8.1 Reliability and stability

Modern LLMs exhibit unpredictable behaviour across queries. Psychometric-style evaluations show that scaling up models and applying instruction fine-tuning improves prompt sensitivity but does not guarantee reliability: larger, more instructable models "tend to give an apparently sensible yet wrong answer" more

often than their smaller predecessors, and their error distribution varies with the phrasing of the prompt [184]. In PenTesting, this manifests as inconsistent command generation and unstable reasoning across runs. Early work pairing GPT-3.5 with a vulnerable VM reported that the agent routinely hallucinated commands that did not exist on the target (e.g. calling a non-existent `exploit.sh`) and sometimes ignored multi-step escalation paths, requiring manual intervention to recover [59]. Subsequent systems mitigate instability by organising prompts into persistent PTTs and using summarisation modules to retain context, yet even recent frameworks such as *Cochise* and *VulnBot* occasionally pursue irrelevant tasks or prematurely declare success [64].

8.2 Context and memory constraints

LLMs process information within a fixed *context window*, which defines the number of tokens they can condition on at once. Any input or conversation history beyond this window is effectively invisible to the model and cannot influence its predictions. For example, a model with an 8,000-token window engaged in a long PrivEsc session will begin to lose the earliest commands and outputs once the interaction exceeds this limit; critical steps such as discovered credentials or misconfigurations may drop out of scope entirely.

This constraint is particularly problematic for autonomous PenTesting, where numerous commands and corresponding outputs must be tracked. Without careful state management, the model rapidly loses sight of earlier actions, leading to repeated enumeration steps or missed escalation paths. Happe et al. demonstrated this effect: GPT-4-turbo's success rate in Linux PrivEsc jumped from 33–50 % to 83 % when the agent was provided with high-level guidance and a longer command history [65].

To mitigate these constraints, frameworks such as PentestGPT and Cochise maintain a persistent PTT that summarises completed subtasks and encodes pending goals [40]. Without such an external memory structure, agents can fall into loops and fail to detect PrivEsc entirely [81]. Even with summarisation, context windows remain a fundamental bottleneck for long engagements; research into RAG [87] attempts to overcome this by storing execution results in external vector databases and dynamically retrieving the most relevant context during inference [81].

8.3 Domain knowledge and reasoning gaps

Although pre-trained LLMs encode general world knowledge, they lack the specialised understanding needed for many offensive security tasks. The models' knowledge cut-off dates and lack of access to proprietary tool documentation make them unaware of newly discovered vulnerabilities and the specific syntax of exploitation frameworks. In a PrivEsc benchmark, adding background information from escalation guides improved success for GPT-3.5 but offered little benefit to GPT-4, suggesting that even advanced models struggle to integrate domain knowledge [65]. Fang et al. similarly found that only frontier

models equipped with function calls, document reading and extended context could autonomously hack websites; when these capabilities were removed, success dropped from 73.3 % to 13 %, and open-source models solved none of the tested vulnerabilities [48]. These results underscore that current LLMs cannot reason reliably about complex infrastructure without external tools and specialised training.

8.4 Hallucinations and command brittleness

Hallucination—the production of convincing yet inaccurate information—is a widely recognised failure mode of LLMs [111]. In offensive security contexts, hallucinations manifest as fabricated command names, invalid syntax, or incorrect vulnerability details. Empirical studies in general NLP show that shaped-up models are more confident yet less prudent; they avoid refusing to answer even when uncertain [184]. In PenTesting this translates into an increased likelihood of the model issuing dangerous commands without validation. Techniques such as CoT prompting and self-refinement can reduce hallucinations, but no method fully eliminates them, necessitating human oversight. Indeed, research suggests that using external knowledge and automated feedback can reduce hallucinations [111, 118].

8.5 Ethical, Safety, Privacy and Governance Challenges

Research on LLM-powered PenTesting systems raises significant ethical, safety, and governance challenges that must be systematically addressed to ensure responsible design, evaluation, and deployment. These systems operate at the intersection of AI and offensive cybersecurity, combining autonomous reasoning with the capability to execute actions that may affect real-world infrastructure. Consequently, they represent a clear example of *dual-use technologies*—tools developed to enhance defensive readiness but which, if misapplied, could facilitate malicious activity [30].

A foundational concern is the **dual-use dilemma**. LLM-augmented agents capable of automated reconnaissance and exploitation can substantially reduce the technical barrier to entry for adversaries, particularly when research artefacts are released without adequate safeguards [169]. Responsible dissemination practices—including coordinated vulnerability disclosure, controlled release of code and prompts, and evaluation within isolated test environments—are therefore essential to balancing scientific transparency with risk mitigation [63]. At the same time, the rapid convergence of academic, commercial, and state-sponsored research initiatives suggests that the pace of innovation in this domain is unlikely to slow. Economic incentives and geopolitical competition further reinforce the inevitability of adversarial experimentation with these technologies [3].

A related challenge concerns the **proliferation and accessibility** of locally executable models. Advances in open-source model distribution and cloud-based infrastructure have significantly lowered the cost of adapting LLMs for specialised tasks, including offensive security workflows. Even when centralised

APIs implement safety filters, containment mechanisms remain limited: once model weights are publicly available, they can be fine-tuned or repurposed outside institutional oversight at relatively low cost [25]. This reality weakens assumptions that AI safety can be enforced solely through platform-level controls and underscores the importance of governance mechanisms that extend beyond technical safeguards.

Another critical dimension is **accountability and operational oversight**. The introduction of autonomous or semi-autonomous agents into offensive testing workflows raises complex questions of liability, particularly if an agent operates beyond authorised scope or produces unintended consequences [55, 79]. Emerging best practices from both academia and industry emphasise maintaining a HITL for high-impact actions, enforcing explicit engagement boundaries, and preserving comprehensive command logs to support traceability and post-incident analysis [9,40]. These practices align with established guidance in NIST SP 800-115, which highlights the importance of governance, authorisation, and documentation in structured PenTesting engagements [147].

Safety and alignment risks further complicate the deployment of LLM-powered systems. Despite recent advances in safety engineering, LLMs remain susceptible to prompt injection attacks and safety filter circumvention, which may enable unintended or unauthorised actions, and large-scale evaluations have documented persistent reliability and safety limitations in state-of-the-art models such as GPT-4 [111]. Experimental evidence demonstrates that relatively minor prompt modifications can alter model behaviour in ways that bypass intended safeguards, particularly when models are deployed without robust execution constraints [59, 116]. Mitigating these risks requires layered controls, including hardened alignment mechanisms, sandboxed execution environments, and continuous monitoring of agent behaviour during runtime.

Privacy and data governance considerations are equally significants [37]. Many LLM-based prototypes rely on cloud-hosted inference services that process sensitive operational data, creating potential risks of data exposure or regulatory non-compliance. Under frameworks such as the European Union's *General Data Protection Regulation* (GDPR), transmitting production artefacts or user-related information to external processing services without appropriate safeguards may violate principles of lawful processing and data minimisation [46]. To address these concerns, recent research advocates the adoption of locally hosted inference pipelines and privacy-preserving system architectures that maintain sensitive information within organisational boundaries.

LLMs are already maliciously exploited in darknet markets, where operators sell customised offensive models such as WormGPT and FraudGPT for social engineering, malware, and payload generation, along with variants like DarkBert, XXXGPT, and WolfGPT [44, 78, 94, 98]. Although their capabilities are unverifiable behind paywalls, their emergence signals accelerating generative-AI weaponisation and the need for proactive mitigation as such tools evolve toward autonomous offensive systems.

These ethical and governance challenges are not unprecedented. Similar debates have historically accompanied the release of widely used offensive security frameworks such as Metasploit, Cobalt Strike, and Sliver C2, which have been employed by both legitimate red teams and malicious actors [36, 86]. The emergence of LLM-powered PenTesting systems can therefore be viewed as an evolution of established dual-use tensions rather than a wholly novel phenomenon.

More broadly, the integration of LLMs into offensive security workflows reflects a structural shift in the cybersecurity landscape. The widespread availability of capable models means that adversaries are increasingly likely to adopt these tools, creating a persistent cycle of technological competition often described as a *Red Queen's race*, in which defenders must continuously innovate to maintain parity with evolving threats [31, 66]. In this context, LLMs simultaneously lower barriers to offensive capability while offering defenders new mechanisms for simulation, detection, and automated response.

Existing regulatory frameworks, including international export-control regimes such as the Wassenaar Arrangement, provide only partial coverage for AI-driven security technologies due to their software-centric and easily distributable nature [151]. Addressing the ethical implications of LLM-powered PenTesting systems is therefore not a peripheral concern but a foundational requirement for responsible research and deployment, as recent studies emphasise that unresolved reliability, accountability, and safety risks continue to constrain the secure operational adoption of LLM-based offensive security tools [64]. Future work must integrate structured governance models, enforceable safety constraints, and transparent evaluation methodologies to ensure that advances in AI-assisted PenTesting strengthen defensive capabilities without inadvertently enabling misuse.

8.6 Computational cost and accessibility

Training and operating frontier LLMs is resource-intensive. The compute cost of training GPT-4 reportedly exceeded US$100 million [29], and executing complex PenTesting agents entails non-trivial API expenses. Fang et al. estimate that autonomously hacking a single website via an LLM agent costs about US$8.8 per attempt [48], a figure that does not include the cost of failed runs. Although this is cheaper than hiring human experts, it remains prohibitively expensive for many practitioners, especially when agents need to engage in lengthy reconnaissance and exploitation cycles. Moreover, the reliance on proprietary models means that open-source alternatives often underperform; in Fang's study, GPT-3.5's success rate dropped to 6.7 % and all open-source models failed entirely [48]. This lack of parity restricts access for researchers and reinforces commercial platform monopolies.

8.7 The Case for HITL PenTesting

As discussed in the preceding subsections, current LLM-driven PenTesting systems remain constrained by reliability limitations, context management challenges, hallucination risks, and unresolved ethical and governance concerns.

These factors collectively make fully autonomous operation difficult to justify in safety-critical environments, particularly when agents are granted network-level access or elevated privileges. Consequently, many recent systems adopt a HITL design to ensure that critical decisions remain subject to human judgement and oversight.

In practice, HITL architectures allow LLMs to support planning, enumeration, and candidate exploit generation, while reserving execution approval, scope validation, and high-impact actions for authorised operators. This approach aligns with established professional guidance for controlled security testing engagements and provides a pragmatic balance between automation efficiency and operational accountability. As a result, HITL configurations currently represent a defensible and widely adopted architectural pattern for deploying LLM-assisted PenTesting systems in real-world settings.

8.8 Summary and outlook

The limitations outlined above reveal why fully autonomous, trustworthy LLM-driven PenTesting remains an open research challenge. While foundation models excel at pattern matching and rapid prototyping, their inconsistent reasoning, limited context windows, hallucination propensity, security vulnerabilities and high operational costs necessitate careful mitigation. Current research therefore focuses on enhancing reliability through structured memory (task trees and summarisation), improving domain reasoning via RAG and fine-tuning, developing red-team methods to harden models against prompt injection, and exploring efficient deployment strategies. Until these challenges are addressed, LLMs should augment rather than replace human PenTesters [4–6, 9, 12, 13].

9 Related Work

This section situates the present study within the existing body of research on LLMs and cybersecurity.

Systematic mappings and surveys of cybersecurity research outline major themes, trends, and gaps across the field while documenting the growing role of LLMs in both defensive and offensive contexts, including threat detection, analysis, automated response, and broader security applications [24,67,134,176]. Subsequent reviews focus specifically on LLM–cybersecurity intersections, proposing taxonomies of applications, system designs, and integration approaches, and examining challenges related to reliability, governance, reproducibility, and evaluation practices [17, 43, 99, 170, 177, 180]. Complementary work examines benchmarking methodologies for LLM-driven offensive security and surveys security and privacy risks to LLMs—including jailbreaks, data poisoning, and information leakage—alongside emerging mitigation strategies [61, 185].

Collectively, prior surveys demonstrate the rapid growth of research at the intersection of LLMs and cybersecurity, documenting applications, risks, and emerging challenges across both defensive and offensive domains. However, the

diversity of system designs, tasks, and evaluation approaches highlights the need for a focused synthesis that examines concrete implementations and operational capabilities of LLM-driven security tools. The present study builds on this foundation by providing a structured analysis of contemporary LLM-aided PenTesting systems, emphasising their architectural characteristics, reasoning mechanisms, and practical performance. In doing so, it offers a consolidated view of the current state of the art and supports clearer comparison and understanding of this evolving research area.

A parallel line of research in automated PenTesting focuses on systems driven by classical ML and reinforcement learning (RL) rather than LLMs. These approaches typically use historical data or environment feedback to support vulnerability prioritisation or adaptive exploit selection, enabling more systematic decision-making during attack execution. For example, *NextGen-PenTest* [57] employs a Random Forest model to predict exploit success probabilities and rank vulnerabilities for targeted assessment, while *DeepExploit* [35] applies RL to automate exploit selection and execution within PenTesting workflows. Collectively, ML/RL-based systems emphasise predictive modelling and adaptive decision-making to improve efficiency and scalability in vulnerability discovery and exploitation, representing an important precursor to contemporary LLM-driven automation. Since these systems are not directly based on LLMs, they fall outside the scope of this survey.

10 Conclusion and Future Work

In this paper, we surveyed the rapid emergence of LLM-powered PenTesting systems from late 2022 to early 2026, spanning advisor-style copilots, semi-autonomous tool-orchestrators, post-exploitation agents, web-focused frameworks, enterprise assumed-breach systems, and dedicated benchmarking platforms. We analysed representative systems in terms of architecture, autonomy, target surface, attack-phase coverage, and evaluation practice, and synthesised recurring design patterns into a practical reference model (planner/generator, executor, memory and summarisation, retrieval, safety/governance, and reporting). Across the literature, most systems concentrate on early-stage exploitation or narrowly scoped tasks, while robust long-horizon operation (context persistence, reliable adaptation, etc.) remains uncommon; consequently, human oversight and constrained execution continue to be the dominant deployment posture.

Future research should prioritise: (i) *more realistic and reproducible evaluation*, including benchmarks that capture enterprise complexity (multi-host state, credential chains, lateral movement, and noisy telemetry) and report success with cost, time, and safety metrics; (ii) *robust memory and state management* (task trees, structured logs, retrieval, and verifiable summaries) to reduce loops and brittle behaviour in long engagements; (iii) *strong governance-by-design*, including scope enforcement, sandboxing, policy-aware tool gateways, and audit-grade logging to mitigate dual-use risk; and (iv) *efficient, accessible deployments*, such as hybrid designs that reserve high-capability reasoning models for planning

while using cheaper/faster models for execution, and improved open-weight baselines to reduce reliance on proprietary APIs. Ultimately, the evidence suggests that near-term progress will come less from 'fully autonomous hacking' and more from well-governed, measurable, and reproducible AI-augmented workflows that reliably amplify expert testers while remaining safe and accountable.

References

1. Abrams, M.D., Jajodia, S.G., Podell, H.J. (eds.): Information Security: An Integrated Collection of Essays. IEEE Computer Society Press, Institute of Electrical and Electronics Engineers, Washington, DC, United States (May 1995), https://dl.acm.org/doi/abs/10.5555/545910
2. Abu-Dabaseh, F., Alshammari, E.: Automated penetration testing: An overview. In: Proceedings of the 4th international conference on natural language computing, Copenhagen, Denmark. pp. 121–129 (2018), https://airccj.org/CSCP/vol8/csit88610.pdf
3. AI Index Steering Committee: Artificial intelligence index report 2025. Tech. rep., Stanford University Human-Centered Artificial Intelligence (HAI) (2025), https://hai.stanford.edu/assets/files/hai_ai_index_report_2025.pdf, accessed 31 July 2025
4. Al-Sinani, H.S., Mitchell, C.J.: AI-augmented ethical hacking: A practical examination of manual exploitation and privilege escalation in Linux environments. CoRR **abs/2411.17539** (Nov 2024). https://doi.org/10.48550/ARXIV.2411.17539, https://doi.org/10.48550/arXiv.2411.17539
5. Al-Sinani, H.S., Mitchell, C.J.: AI-generated papers and manipulation of academic metrics: A case study. In: Proceedings of the 2nd International Workshop on Cybersecurity: Blockchain and Artificial Intelligence Applications (CyBAI'25), part of the 8th Congress on Information Science and Technology (CiSt), 4–10 October 2025. IEEE, Marrakech, Morocco (Oct 2025), https://doi.org/10.48550/arXiv.2503.23414, to appear. A preliminary version was posted on arXiv in March 2025 as arXiv:2503.23414 [cs.CR]
6. Al-Sinani, H.S., Mitchell, C.J.: Introducing PenTest++: an AI-augmented, automated, ethical hacking system. In: Proceedings of the 2nd International Workshop on Cybersecurity: Blockchain and Artificial Intelligence Applications (CyBAI '25), part of the 8th Congress on Information Science and Technology (CiSt), 4–10 October 2025. pp. 565–570. IEEE, Marrakech, Morocco (Oct 2025), https://ieeexplore.ieee.org/document/11224070, a preliminary version was posted on arXiv on February 13th, 2025 as arXiv:2502.09484 [cs.CR]
7. Al-Sinani, H.S., Mitchell, C.J.: PenTest++: Elevating ethical hacking with AI and automation. CoRR **abs/2502.09484** (Feb 2025). https://doi.org/10.48550/ARXIV.2502.09484, https://doi.org/10.48550/arXiv.2502.09484
8. Al-Sinani, H.S., Mitchell, C.J.: PenTest2.0: Towards autonomous privilege escalation using GenAI. CoRR **abs/2507.06742** (jul 2025). https://doi.org/10.48550/ARXIV.2507.06742, https://doi.org/10.48550/arXiv.2507.06742
9. Al-Sinani, H.S., Mitchell, C.J.: PenTest2.0: Towards autonomous privilege escalation using GenAI. CoRR **abs/2507.06742** (Jul 2025). https://doi.org/10.48550/arXiv.2507.06742, https://arxiv.org/abs/2507.06742

10. Al-Sinani, H.S., Mitchell, C.J.: WiFiPenTester: Advancing wireless ethical hacking with governed GenAI. CoRR **abs/2601.23092** (Jan 2026). https://doi.org/10.48550/arXiv.2601.23092, https://arxiv.org/abs/2601.23092
11. Al-Sinani, H.S., Mitchell, C.J., Al-Hosni, A.S.: PenTest2.0: Advancing ethical hacking with GenAI-driven privilege escalation. In: Proceedings of the 40th International Conference on Advanced Information Networking and Applications (AINA '26), Wellington, New Zealand, April 8–10, 2026. Lecture Notes in Data Engineering and Communication Technologies (LNDECT), Springer (Apr 2026), to appear
12. Al-Sinani, H.S., Mitchell, C.J., Sahli, N., Al-Siyabi, M.: Unleashing AI in ethical hacking. In: Martinelli, F., Rios, R. (eds.) Proceedings of Security and Trust Management — 20th International Workshop, STM '24, Bydgoszcz, Poland, September 19–20, 2024. LNCS: Lecture Notes in Computer Science, vol. 15235, pp. 140–151. Springer (2024). https://doi.org/10.1007/978-3-031-76371-7_10, https://link.springer.com/chapter/10.1007/978-3-031-76371-7_10
13. Al-Sinani, H.S., Sahli, N., Mitchell, C.J., Al-Siyabi, M.: Advancing ethical hacking with AI: A Linux-based experimental study. In: Costa, G., Montanari, R., Carminati, M., Sciarretta, G. (eds.) Proceedings of the Joint National Conference on Cybersecurity (ITASEC & SERICS '25), February 03–08, 2025, Bologna, Italy. vol. 3962. CEUR-WS (Feb 2025). https://doi.org/10.1007/978-3-031-76371-7_10, https://ceur-ws.org/Vol-3962/paper7.pdf
14. Alford, R., Lawrence, D., Kouremetis, M.: Caldera: A red-blue cyber operations automation platform. https://caldera.mitre.org/ (2022)
15. Alibaba Group: Qwen3: A family of large reasoning models. https://github.com/QwenLM (2024), accessed January 2026
16. Almazrouei, E., Alobeidli, H., Alshamsi, A., Cappelli, A., et al.: The Falcon series of open language models. CoRR **abs/2311.16867** (Nov 2023). https://doi.org/10.48550/arXiv.2311.16867, https://arxiv.org/abs/2311.16867, submitted on 28 Nov 2023
17. Alqahtani, H., Kumar, G.: A comprehensive review of generative AI techniques and their impact on cybersecurity. Soft Computing **29**(13), 4945–4982 (Aug 2025). https://doi.org/10.1007/s00500-025-10071-x
18. AlShebli, H., Beheshti, B.D.: A study on penetration testing process and tools. In: 2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT). pp. 1–7. IEEE, Farmingdale, NY, USA (2018). https://doi.org/10.1109/LISAT.2018.8378035, https://doi.org/10.1109/LISAT.2018.8378035
19. Alshehri, I., Alshehri, A., Almalki, A., Bamardouf, M., Akbar, A.: BreachSeek: a multi-agent automated penetration tester. CoRR **abs/2409.03789** (Sep 2024). https://doi.org/10.48550/arXiv.2409.03789
20. Amershi, S., Cakmak, M., Knox, W.B., Kulesza, T.: Power to the people: The role of humans in interactive machine learning. In: AI Magazine. vol. 35, pp. 105–120 (2014). https://doi.org/10.1609/aimag.v35i4.2513, https://ojs.aaai.org/index.php/aimagazine/article/view/2513
21. Anderson, J.P.: Computer security technology planning study. Tech. Rep. ESD-TR-73-51-VOL-1, James P. Anderson and Co., Fort Washington, PA, USA (Oct 1972), https://apps.dtic.mil/sti/html/tr/AD0758206/, technical Re-

port prepared for the U.S. Air Force Electronic Systems Division; Approved for public release
22. Anthropic: Introducing the next generation of claude. Official model announcement (Mar 2024), https://www.anthropic.com/news/claude-3-family, announcement of the Claude 3 model family (Haiku, Sonnet, Opus)
23. Arkin, B., Stender, S., McGraw, G.: Software penetration testing. IEEE Security & Privacy **3**(1), 84–87 (2005). https://doi.org/10.1109/MSP.2005.23
24. Aydos, M., Aldan, Ç., Coşkun, E., Soydan, A.: Security testing of web applications: A systematic mapping of the literature. Journal of King Saud University – Computer and Information Sciences **34**(9), 6775–6792 (Oct 2022). https://doi.org/10.1016/j.jksuci.2021.05.017
25. Beeching, E., Belkada, Y., Rasul, K., Tunstall, L., von Werra, L., Rajani, N., Lambert, N.: StackLLaMA: An RL fine-tuned LLaMA model for stack exchange question answering. https://huggingface.co/trl-lib/llama-7b-se-rl-peft (2023). https://doi.org/10.57967/hf/0513, model card and code release only; no formal peer-reviewed publication. Accessed 31 July 2025
26. Bishop, M.: About penetration testing. IEEE Security & Privacy **5**(6), 84–87 (Dec 2007). https://doi.org/10.1109/MSP.2007.159
27. Boiko, D.A., MacKnight, R., Gomes, G.: Emergent autonomous scientific research capabilities of large language models. CoRR **abs/2304.05332** (Apr 2023). https://doi.org/10.48550/arXiv.2304.05332, https://arxiv.org/abs/2304.05332
28. Bran, A.M., Cox, S., Schilter, O., Baldassari, C., White, A.D., Schwaller, P.: Augmenting large language models with chemistry tools. Nature Machine Intelligence **6**(5), 525–535 (May 2024). https://doi.org/10.1038/s42256-024-00832-8, https://doi.org/10.1038/s42256-024-00832-8
29. Brown, T.B., Mann, B., Ryder, N., et al.: Language Models are Few-Shot Learners. In: Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Virtual Conference, 6–12 December 2020. Advances in Neural Information Processing Systems, vol. 33, pp. 1877–1901. Curran Associates, Inc. (2020). https://doi.org/10.48550/arXiv.2005.14165
30. Brundage, M., Avin, S., Clark, J., Toner, H., Eckersley, P., Garfinkel, B., Dafoe, A., Scharre, P., Zeitzoff, T., Filar, B., Anderson, H., Roff, H., Allen, G.C., Steinhardt, J., Flynn, C., Ó hÉigeartaigh, S., Beard, S., Belfield, H., Farquhar, S., Lyle, C., Crootof, R., Evans, O., Page, M., Bryson, J., Yampolskiy, R., Amodei, D.: The malicious use of artificial intelligence: Forecasting, prevention, and mitigation. CoRR **abs/1802.07228** (Dec 2024). https://doi.org/10.48550/arXiv.1802.07228, https://arxiv.org/abs/1802.07228, submitted on 20 Feb 2018 (v1), last revised 1 Dec 2024 (v2)
31. Bukac, V., Lorenc, V., Matyáš, V.: Red queen's race: APT win-win game. In: Security Protocols XXII: 22nd International Workshop, Cambridge, UK, March 19–21, 2014, Revised Selected Papers. LNCS: Lecture Notes in Computer Science, vol. 8809, pp. 55–61. Springer (2014). https://doi.org/10.1007/978-3-662-44774-1_8, https://doi.org/10.1007/978-3-662-44774-1_8
32. Chase, H.: LangChain. Software framework for building LLM-powered applications (Oct 2022), https://www.langchain.com/, Repository: https://github.com/langchain-ai/langchain. Licensed under MIT License.

33. Cohere: Command r+ (cohere). Product documentation (2024), https://docs.cohere.com/docs/command-r-plus, cohere's Retrieval-Augmented Generation model for long-context and multi-step tasks.
34. Conover, M., Hayes, M., Mathur, A., Xie, J., Wan, J., Shah, S., Ghodsi, A., Wendell, P., Zaharia, M., Xin, R.: Dolly 2: Free dolly: Introducing the world's first truly open instruction-tuned LLM. https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm (2023), software project and blog announcement only; no formal publication available. Accessed 30 July 2025
35. CQR: AI in penetration testing: Exploitation phase. Online (May 2023), https://cqr.company/blog/ai-in-penetration-testing-exploitation-phase/, accessed on 27 March 2026
36. Cybereason Global SOC and Incident Response Team: Sliver C2 leveraged by many threat actors. https://www.cybereason.com/blog/sliver-c2-leveraged-by-many-threat-actors (Jan 2023), accessed 26 July 2025
37. Das, B.C., Amini, M.H., Wu, Y.: Security and privacy challenges of large language models: A survey. ACM Computing Surveys **57**(6) (Feb 2025). https://doi.org/10.1145/3712001, https://dl.acm.org/doi/abs/10.1145/3712001
38. De Pasquale, G., Grishchenko, I., Iesari, R., Pizarro, G., Cavallaro, L., Kruegel, C., Vigna, G.: ChainReactor: Automated privilege escalation chain discovery via ai planning. In: Proceedings of the 33rd USENIX Security Symposium (USENIX Security 24). pp. 5913–5929. USENIX Association, Philadelphia, PA, USA (2024), https://www.usenix.org/conference/usenixsecurity24/presentation/de-pasquale
39. DeepSeek-AI, Liu, A., Feng, B., Xue, B., Wang, B., et al.: Deepseek-v3 technical report. CoRR **abs/2412.19437** (Feb 2025). https://doi.org/10.48550/arXiv.2412.19437, https://arxiv.org/abs/2412.19437, submitted on 27 Dec 2024 (v1), last revised 18 Feb 2025 (v2)
40. Deng, G., Liu, Y., Mayoral-Vilches, V., Liu, P., Li, Y., Xu, Y., Zhang, T., Liu, Y., Pinzger, M., Rass, S.: PentestGPT: Evaluating and harnessing Large Language Models for automated penetration testing. In: Proceedings of the 33rd USENIX Security Symposium (USENIX Security '24). pp. 847–864. USENIX Association, Philadelphia, PA, USA (Aug 2024), https://www.usenix.org/conference/usenixsecurity24/presentation/deng
41. Denny, P., Kumar, V., Giacaman, N.: Conversing with copilot: Exploring prompt engineering for solving CS1 problems using natural language. In: Proceedings of the 54th ACM Technical Symposium on Computer Science Education. pp. 1136–1142 (2023). https://doi.org/10.1145/3545945.3569755, https://doi.org/10.1145/3545945.3569755
42. Dong, Q., Li, L., Dai, D., Zheng, C., Ma, J., Li, R., Xia, H., Xu, J., Wu, Z., Liu, T., Chang, B., Sun, X., Li, L., Sui, Z.: A survey on in-context learning. CoRR **abs/2301.00234** (Oct 2024). https://doi.org/10.48550/arXiv.2301.00234, https://arxiv.org/abs/2301.00234, submitted on 31 Dec 2022 (v1), last revised 5 Oct 2024 (this version, v6)
43. Dube, R.: Large language models in information security research: A January 2024 survey. ResearchGate Preprint (Jan 2024)

44. Dutta, T.S.: Hackers released new black hat AI tools XXXGPT and WolfGPT. https://cybersecuritynews.com/black-hat-ai-tools-xxxgpt-and-wolf-gpt/ (Aug 2023), accessed 26 July 2025
45. Engebretson, P.: The Basics of Hacking and Penetration Testing: Ethical Hacking and Penetration Testing Made Easy. Syngress / Elsevier, Waltham, MA, USA, 2nd edn. (Aug 2013), 225 pages
46. European Parliament and Council of the European Union: Regulation (EU) 2016/679 of the european parliament and of the council on the protection of natural persons with regard to the processing of personal data and on the free movement of such data (general data protection regulation). Official Journal of the European Union, L119 (Apr 2016), https://eur-lex.europa.eu/eli/reg/2016/679/oj, accessed 26 July 2025
47. Fang, R., Bindu, R., Gupta, A., Kang, D.: LLM agents can autonomously exploit one-day vulnerabilities. CoRR **abs/2404.08144** (Apr 2024). https://doi.org/10.48550/arXiv.2404.08144, https://arxiv.org/abs/2404.08144
48. Fang, R., Bindu, R., Gupta, A., Zhan, Q., Kang, D.: LLM agents can autonomously hack websites. CoRR **abs/2402.06664** (Feb 2024). https://doi.org/10.48550/arXiv.2402.06664, https://arxiv.org/abs/2402.06664
49. Fatima, A., Khan, T.A., Abdellatif, T.M., Zulfiqar, S., Asif, M., Safi, W., Hamadi, H.A., Al-Kassem, A.H.: Impact and research challenges of penetration testing and vulnerability assessment on network threat. In: 2023 International Conference on Business Analytics for Technology and Security (ICBATS). pp. 1–8. IEEE, Institute of Electrical and Electronics Engineers (2023). https://doi.org/10.1109/ICBATS57792.2023.10111168
50. Geer, D., Harthorne, J.: Penetration testing: A duet. In: 18th Annual Computer Security Applications Conference (ACSAC 2002). pp. 185–195. IEEE, Institute of Electrical and Electronics Engineers (2002). https://doi.org/10.1109/CSAC.2002.1176290
51. Gemini Team, Anil, R., Borgeaud, S., Alayrac, J.B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A.M., Hauth, A., Millican, K., Silver, D., Johnson, M., Antonoglou, I., Schrittwieser, J., Glaese, A., et al.: Gemini: A family of highly capable multimodal models. CoRR **abs/2312.11805** (May 2025). https://doi.org/10.48550/arXiv.2312.11805, https://arxiv.org/abs/2312.11805, submitted on 19 Dec 2023 (v1), last revised 9 May 2025 (this version, v5)
52. Geng, X., Gudibande, A., Liu, H., Wallace, E., Abbeel, P., Levine, S., Song, D.: Koala: A dialogue model for academic research. https://bair.berkeley.edu/blog/2023/04/03/koala/ (2023), software project and blog announcement only; no formal publication available. Accessed 30 July 2025
53. Gerganov, G.: llama.cpp: Inference of llama model in pure c/c++. https://github.com/ggerganov/llama.cpp (2023), accessed July 2025
54. Gioacchini, L., Delsanto, A., Drago, I., Mellia, M., Siracusano, G., Bifulco, R.: AutoPenBench: A vulnerability testing benchmark for generative agents. In: Potdar, S., Rojas-Barahona, L., Montella, S. (eds.) Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track (EMNLP 2025), Suzhou, China, 4–9 November 2025. pp. 1615–1624. ACL: Association for Computational Linguistics (2025).

https://doi.org/10.18653/v1/2025.emnlp-industry.114, https://aclanthology.org/2025.emnlp-industry.114/
55. Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., Fritz, M.: Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. CoRR **abs/2302.12173** (Feb 2023). https://doi.org/10.48550/arXiv.2302.12173, https://arxiv.org/abs/2302.12173
56. Guo, D., Yang, D., Zhang, H., et al.: DeepSeek-R incentivizes reasoning in llms through reinforcement learning. Nature **645**(8081), 633–638 (Sep 2025). https://doi.org/10.1038/s41586-025-09422-z, https://doi.org/10.1038/s41586-025-09422-z
57. Hadi, M.H., Al-Saedi, K.H.: Enhancing penetration testing: Leveraging machine learning for ethical hacking. In: Proceedings of the Second International Conference on Innovations of Intelligent Informatics, Networking, and Cybersecurity (3INC 2024): Babylon, Iraq, October 15–16, 2024. pp. 230–248. Lecture Notes in Networks and Systems (LNNS), Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-031-81065-7_15
58. Handa, A., Sharma, A., Shukla, S.K.: Machine learning in cybersecurity: A review. WIREs Data Mining and Knowledge Discovery **9**(4), e1306 (Jul 2019). https://doi.org/10.1002/widm.1306, https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1306
59. Happe, A., Cito, J.: Getting pwn'd by AI: Penetration testing with Large Language Models. In: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23). pp. 2082–2086. ESEC/FSE, ACM: Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3611643.3613083, https://dl.acm.org/doi/abs/10.1145/3611643.3613083
60. Happe, A., Cito, J.: Understanding hackers' work: An empirical study of offensive security practitioners. In: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 1669–1680. ESEC/FSE 2023, ACM: Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3611643.3613900, https://doi.org/10.1145/3611643.3613900
61. Happe, A., Cito, J.: Benchmarking practices in LLM-driven offensive security: Testbeds, metrics, and experiment design. CoRR **abs/2504.10112** (Jun 2025). https://doi.org/10.48550/arXiv.2504.10112, https://arxiv.org/abs/2504.10112
62. Happe, A., Cito, J.: Can LLMs hack enterprise networks? Autonomous assumed breach Penetration-Testing active directory networks. ACM Transactions on Software Engineering and Methodology (Sep 2025). https://doi.org/10.1145/3766895
63. Happe, A., Cito, J.: On the ethics of using LLMs for offensive security. CoRR **abs/2506.08693** (Jun 2025). https://doi.org/10.48550/arXiv.2506.08693, https://arxiv.org/abs/2506.08693
64. Happe, A., Cito, J.: On the surprising efficacy of LLMs for penetration-testing. CoRR **abs/2507.00829** (Jul 2025). https://doi.org/10.48550/arXiv.2507.00829, https://arxiv.org/abs/2507.00829
65. Happe, A., Kaplan, A., Cito, J.: LLMs as hackers: Autonomous linux privilege escalation attacks. Empirical Software Engineering **31**(70) (Feb 2026). https://doi.org/10.1007/s10664-025-10758-3

66. Harang, R., Ducau, F.N.: Measuring the speed of the red queen's race. In: Black Hat USA 2018. Las Vegas, NV, USA (Aug 2018), https://www.blackhat.com/us-18/briefings/schedule/#measuring-the-speed-of-the-red-queens-race-11040
67. Hassanin, M., Moustafa, N.: A comprehensive overview of large language models (LLMs) for cyber defences: Opportunities and directions. CoRR **abs/2405.14487** (May 2024). https://doi.org/10.48550/arXiv.2405.14487
68. Hazell, J.: Spear phishing with large language models. CoRR **abs/2305.06972** (May 2023). https://doi.org/10.48550/arXiv.2305.06972, https://arxiv.org/abs/2305.06972
69. Herzog, P., et al.: Open Source Security Testing Methodology Manual (OSSTMM), Version 3: Contemporary Security Testing and Analysis. Institute for Security and Open Methodologies (ISECOM) (2020), https://www.isecom.org/OSSTMM.3.pdf, accessed 29 July 2025
70. Hilario, E., Azam, S., Sundaram, J., Imran Mohammed, K., Shanmugam, B.: Generative AI for PenTesting: The good, the bad, the ugly. International Journal of Information Security **23**(3), 2075–2097 (Jun 2024). https://doi.org/10.1007/s10207-024-00835-x, https://doi.org/10.1007/s10207-024-00835-x
71. Huang, H., Shi, J., Chen, J., Zhang, T., Li, Y., Yang, C., Ouh, E.L., Shar, L.K., Lo, D.: PenForge: On-the-fly expert agent construction for automated penetration testing. In: Proceedings of the 48th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER '26), April 12–18, 2026, Rio de Janeiro, Brazil — to appear. pp. 1–5. ICSE-NIER '26, ACM – Association for Computing Machinery, New York, NY, USA (Apr 2026), https://arxiv.org/abs/2601.06910
72. Huang, J., Zhu, Q.: PenHeal: A two-stage LLM framework for automated pentesting and optimal remediation. In: Proceedings of the Workshop on Autonomous Cybersecurity. pp. 11–22. AutonomousCyber '24, ACM: Association for Computing Machinery, New York, NY, USA (Nov 2024). https://doi.org/10.1145/3689933.3690831, https://doi.org/10.1145/3689933.3690831
73. Hutchins, E.M., Cloppert, M.J., Amin, R.M.: Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. Leading Issues in Information Warfare & Security Research **1**(1), 80–106 (2011), https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/LM-White-Paper-Intel-Driven-Defense.pdf
74. Infosec Institute: The history of penetration testing. https://www.infosecinstitute.com/resources/penetration-testing/the-history-of-penetration-testing/ (Jul 2019), accessed 24 July 2025
75. Isozaki, I., Shrestha, M., Console, R., Kim, E.: Towards automated penetration testing: Introducing LLM benchmark, analysis, and improvements. In: Adjunct Proceedings of the 33rd ACM Conference on User Modeling, Adaptation and Personalization (UMAP Adjunct 2025), June 16 – 19, 2025, New York City, NY, USA. pp. 404–419. ACM: Association for Computing Machinery (2025). https://doi.org/10.1145/3708319.3733804
76. Jiang, A.Q., Sablayrolles, A., Roux, A., Mensch, A., et al.: Mixtral of experts. CoRR **abs/2401.04088** (Jan 2024). https://doi.org/10.48550/arXiv.2401.04088, https://arxiv.org/abs/2401.04088

77. Jimenez, C.E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., Narasimhan, K.: SWE-bench: Can language models resolve real-world GitHub issues? In: Proceedings of the Twelfth International Conference on Learning Representations. ICLR '24, Vienna, Austria (May 2024). https://doi.org/10.48550/arXiv.2310.06770, https://arxiv.org/abs/2310.06770, data, code, and leaderboard available at https://www.swebench.com
78. Jin, Y., Jang, E., Cui, J., Chung, J.W., Lee, Y., Shin, S.: DarkBERT: A language model for the dark side of the internet. CoRR **abs/2305.08596** (May 2023). https://doi.org/10.48550/arXiv.2305.08596, https://arxiv.org/abs/2305.08596
79. Kang, D., Li, X., Stoica, I., Guestrin, C., Zaharia, M., Hashimoto, T.: Exploiting programmatic behavior of LLMs: Dual-use through standard security attacks. In: Proceedings of the 2024 IEEE Security and Privacy Workshops (SPW). pp. 132–143. IEEE, San Francisco, CA, USA (May 2024). https://doi.org/10.1109/SPW63631.2024.00018
80. Kennedy, D., Aharoni, M., Kearns, D., O'Gorman, J., Graham, D.G.: Metasploit: The Penetration Tester's Guide. No Starch Press, San Francisco, CA, USA, 2 edn. (Jan 2025), https://nostarch.com/Metasploit2E
81. Kobayashi, M., Fuchi, M., Zanashir, A., Yoneda, T., Takagi, T.: Construction and evaluation of LLM-based agents for semi-autonomous penetration testing. CoRR **abs/2502.15506** (Feb 2025). https://doi.org/10.48550/arXiv.2502.15506, https://arxiv.org/abs/2502.15506
82. Kojima, T., Gu, S.S., Reid, M., Matsuo, Y., Iwasawa, Y.: Large language models are zero-shot reasoners. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (eds.) Advances in Neural Information Processing Systems (NeurIPS 2022). vol. 35, pp. 22199–22213. Curran Associates, Inc. (2022), https://proceedings.neurips.cc/paper_files/paper/2022/file/8bb0d291acd4acf06ef112099c16f326-Paper-Conference.pdf
83. Kong, H., Hu, D., Ge, J., Li, L., Li, T., Wu, B.: VulnBot: autonomous penetration testing for a multi-agent collaborative framework. CoRR **abs/2501.13411** (Jan 2025). https://doi.org/10.48550/arXiv.2501.13411
84. Kowira, E.M., Nik Suki, N., Nathan, Y.: Automated privilege escalation enumeration and execution script for linux. AIP Conference Proceedings **2802**(1), 150009 (Jan 2024). https://doi.org/10.1063/5.0183425, https://doi.org/10.1063/5.0183425
85. Kumar, A.: Penetration testing statistics, vulnerabilities and trends in 2026 (Oct 2025), https://thecyphere.com/blog/penetration-testing-statistics/, accessed on 25 March 2026
86. Larson, S., Blackford, D.: Cobalt strike: Favorite tool from APT to crimeware. https://www.proofpoint.com/us/blog/threat-insight/cobalt-strike-favorite-tool-apt-crimeware (Mar 2021), accessed 31 July 2025
87. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Kuksa, P., Polosukhin, I., Riedel, S., Kiela, D., et al.: Retrieval-augmented generation for knowledge-intensive NLP tasks. In: Advances in Neural Information Processing Systems (NeurIPS 2020). vol. 33, pp. 9459–9474. Curran Associates, Inc. (Dec 2020), https://papers.nips.cc/paper_files/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html
88. Li, X., Yu, Z., Zhang, Z., Chen, X., Zhang, Z., Zhuang, Y., Sadagopan, N., Beniwal, A.: When thinking fails: The pitfalls of reasoning for instruction-following

in llms. arXiv preprint arXiv:2505.11423 (2025), https://arxiv.org/abs/2505.11423
89. Li, Z., Calvo-Bartolomé, L., Hoyle, A., Xu, P., Dima, A., Fung, J.F., Boyd-Graber, J.: Large language models struggle to describe the haystack without human help: Human-in-the-loop evaluation of topic models (2025), https://arxiv.org/abs/2502.14748
90. Lohn, A.J., Jackson, K.A.: Will AI make cyber swords or shields: A few mathematical models of technological progress. CoRR **abs/2207.13825** (Jul 2022). https://doi.org/10.48550/arXiv.2207.13825, https://arxiv.org/abs/2207.13825, technical companion to the CSET report titled "Will AI Make Cyber Swords or Shields: Using Models to Project the Impact of Technology Development"
91. Mahajan, A.: Burp Suite Essentials. Packt Publishing, 1st edn. (Nov 2014), https://www.packtpub.com/product/burp-suite-essentials/9781783550128
92. Mandiant (FireEye): Apt and the attack lifecycle (2013), https://cs.brown.edu/courses/cs180/static/files/lectures/readings/lecture15/mandiant_apt_excerpt.pdf, accessed 04 August 2025
93. Mandiant (now part of Google Cloud): Targeted attack life cycle (2023), https://cloud.google.com/security/resources/insights/targeted-attack-lifecycle, accessed 04 August 2025
94. Mascellino, A.: AI tool wormgpt enables convincing fake emails for BEC attacks. https://www.infosecurity-magazine.com/news/wormgpt-fake-emails-bec-attacks/ (Jul 2023), accessed 02 August 2025
95. Mialon, G., Dessì, R., Lomeli, M., Nalmpantis, C., Pasunuru, R., Raileanu, R., Rozière, B., Schick, T., Dwivedi-Yu, J., Celikyilmaz, A., Grave, E., LeCun, Y., Scialom, T.: Augmented language models: A survey. CoRR **abs/2302.07842** (Feb 2023). https://doi.org/10.48550/arXiv.2302.07842, https://arxiv.org/abs/2302.07842
96. Mistral AI: Mistral-7b instruct v0.2. Hugging Face Model Card (Dec 2023), https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2, instruction-tuned variant of Mistral-7B-v0.2 with enhanced chat capabilities; released under Apache 2.0 license with open weights. Optimised for practical assistant-like usage.
97. MITRE Corporation: ATT&CK Matrix for Enterprise. https://attack.mitre.org/ (2025), accessed 04 August 2025
98. Montalbano, E.: Darkbert: GPT-based malware trains up on the entire dark web. https://www.darkreading.com/application-security/gpt-based-malware-trains-dark-web (Jun 2023), accessed 02 August 2025
99. Motlagh, F.N., Hajizadeh, M., Majd, M., Najafi, P., Cheng, F., Meinel, C.: Large language models in cybersecurity: State-of-the-art. CoRR **abs/2402.00891** (Feb 2024). https://doi.org/10.48550/arXiv.2402.00891
100. Muzsai, L., Imolai, D., Lukács, A.: Hacksynth: LLM agent and evaluation framework for autonomous penetration testing. CoRR **abs/2412.01778** (Dec 2024). https://doi.org/10.48550/arXiv.2412.01778, https://doi.org/10.48550/arXiv.2412.01778
101. Naik, N., Jenkins, P., Grace, P., Song, J.: Comparing attack models for IT systems: Lockheed Martin's cyber kill chain, MITRE ATT&CK framework

and diamond model. In: Proceedings of the 2022 IEEE International Symposium on Systems Engineering (ISSE). pp. 1–7. IEEE — Institute of Electrical and Electronics Engineers (2022). https://doi.org/10.1109/ISSE54508.2022.10005490, https://ieeexplore.ieee.org/abstract/document/10005490
102. Nakajima, Y.: BabyAGI (2023), https://github.com/yoheinakajima/babyagi, software project only; no formal publication available. Accessed 26 July 2025
103. Nakatani, S.: RapidPen: fully automated IP-to-Shell penetration testing with LLM-based agents. CoRR **abs/2502.16730** (Feb 2025). https://doi.org/10.48550/arXiv.2502.16730
104. NousResearch: Nous-Hermes-2 - Yi-34B. Hugging Face Model Card (Dec 2023), https://huggingface.co/NousResearch/Nous-Hermes-2-Yi-34B
105. Open Information Systems Security Group (OISSG): Information Systems Security Assessment Framework (ISSAF), Draft 0.2 (2006), https://untrustednetwork.net/files/issaf0.2.1.pdf, framework draft freely mirrored; originally published by OISSG circa 2006. Accessed 26 July 2025
106. OpenAI: Introducing ChatGPT. https://openai.com/blog/chatgpt (Nov 2022), https://openai.com/blog/chatgpt, accessed 07 August 2025. ChatGPT, released in November 2022, was powered by GPT-3.5
107. OpenAI: Introducing openai o1: Reasoning models. https://openai.com/index/introducing-openai-o1-preview/ (2024), accessed January 2026
108. OpenAI: Reasoning models: Best practices. https://platform.openai.com/docs/guides/reasoning-best-practices (2024), accessed January 2026
109. OpenAI: Assistants API (v2). https://platform.openai.com/docs/assistants/overview (2025), Accessed: August 11, 2025; Enables creation of AI assistants with tool use, persistent threads, and function calling
110. OpenAI: Introducing gpt-5. https://openai.com/index/introducing-gpt-5 (2025)
111. OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., et al.: GPT-4 technical report. CoRR **abs/2303.08774** (Mar 2024). https://doi.org/10.48550/arXiv.2303.08774, https://arxiv.org/abs/2303.08774, submitted on 15 Mar 2023 (v1), last revised 4 Mar 2024 (v6)
112. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P.F., Leike, J., Lowe, R.: Training language models to follow instructions with human feedback. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (eds.) Advances in Neural Information Processing Systems (NeurIPS 2022). vol. 35, pp. 27730–27744. Curran Associates, Inc. (2022), https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf
113. OWASP Foundation: OWASP Top Ten: The Ten Most Critical Web Application Security Risks (2021), https://owasp.org/www-project-top-ten/, accessed 26 July 2025
114. Pa Pa, Y.M., Tanizaki, S., Kou, T., van Eeten, M., Yoshioka, K., Matsumoto, T.: An attacker's dream? Exploring the capabilities of ChatGPT for developing malware. In: Proceedings of the 16th Cyber Security Experimentation and Test Workshop (CSET '23). pp. 10–18. CSET '23, ACM: Association for Computing Machinery, New York, NY, USA (Aug 2023). https://doi.org/10.1145/3607505.3607513, https://doi.org/10.1145/3607505.3607513

115. Parisi, A., Zhao, Y., Fiedel, N.: TALM: Tool augmented language models. CoRR **abs/2205.12255** (May 2022). https://doi.org/10.48550/arXiv.2205.12255, https://arxiv.org/abs/2205.12255
116. Pathade, C.: Red teaming the mind of the machine: A systematic evaluation of prompt injection and jailbreak vulnerabilities in LLMs. CoRR **abs/2505.04806** (May 2025). https://doi.org/10.48550/arXiv.2505.04806, https://arxiv.org/abs/2505.04806, submitted on 7 May 2025 (v1), last revised 13 May 2025 (v2)
117. PCI Security Standards Council: Penetration Testing Guidance (2017), https://www.pcisecuritystandards.org, accessed 26 July 2025
118. Peng, B., Galley, M., He, P., Cheng, H., Xie, Y., Hu, Y., Huang, Q., Liden, L., Yu, Z., Chen, W., Gao, J.: Check your facts and try again: Improving large language models with external knowledge and automated feedback. CoRR **abs/2302.12813** (Feb 2023). https://doi.org/10.48550/arXiv.2302.12813, https://arxiv.org/abs/2302.12813
119. Phuong, M., Aitchison, M., Catt, E., Cogan, S., et al.: Evaluating frontier models for dangerous capabilities. CoRR **abs/2403.13793** (Mar 2024). https://doi.org/10.48550/arXiv.2403.13793, https://arxiv.org/abs/2403.13793
120. Pols, P.: The Unified Kill Chain: Designing a Unified Kill Chain for Analyzing, Comparing and Defending Against Cyber Attacks. Master's thesis, Cyber Security Academy, The Hague University of Applied Sciences (2017), https://www.unifiedkillchain.com/assets/The-Unified-Kill-Chain.pdf, accessed 26 July 2025
121. Proofpoint: What is the cyber kill chain? definition & steps. Threat Reference, Proofpoint (2025), https://www.proofpoint.com/us/threat-reference/cyber-kill-chain, accessed 24 July 2025
122. PTES Working Group: Penetration testing execution standard (PTES). http://www.pentest-standard.org/ (2014), community-maintained document. Last edited 14 January 2017. Accessed 26 July 2025
123. PTF Project: Penetration Testing Framework, Version 0.59 (2013), http://www.vulnerabilityassessment.co.uk/Penetration%20Test.html, accessed 26 July 2025
124. Qi, X., Zeng, Y., Xie, T., Chen, P.Y., Jia, R., Mittal, P., Henderson, P.: Fine-tuning aligned language models compromises safety, even when users do not intend to! In: Proceedings of the International Conference on Learning Representations (ICLR). International Conference on Learning Representations, Vienna, Austria (May 2024). https://doi.org/10.48550/arXiv.2310.03693, oral presentation. An earlier version is available as an arXiv preprint: https://arxiv.org/abs/2310.03693
125. Raman, R., Calyam, P., Achuthan, K.: ChatGPT or Bard: Who is a better certified ethical hacker? Computers & Security **140**, 103804 (May 2024). https://doi.org/10.1016/j.cose.2024.103804, https://www.sciencedirect.com/science/article/pii/S0167404824001056
126. Rani, N., Milner, K., Shao, M., Udeshi, M., Xi, H., Putrevu, V.S.C., Aggarwal, S., Shukla, S.K., Krishnamurthy, P., Khorrami, F., Shafique, M., Karri, R.: Cyberexplorer: Benchmarking LLM offensive security capabilities in a real-world attacking simulation environment. CoRR **abs/2602.08023** (Feb 2026). https://doi.org/10.48550/arXiv.2602.08023

127. Regina, M., Meyer, M., Goutal, S.: Text data augmentation: Towards better detection of spear-phishing emails. CoRR **abs/2007.02033** (Jul 2021). https://doi.org/10.48550/arXiv.2007.02033, https://arxiv.org/abs/2007.02033, submitted on 4 Jul 2020, last revised 13 Jan 2021 (v3)
128. Sarraute, C., Buffet, O., Hoffmann, J.: Pomdps make better hackers: Accounting for uncertainty in penetration testing. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 26, pp. 1816–1824. AAAI: Association for the Advancement of Artificial Intelligence (2012)
129. Sarraute, C., Buffet, O., Hoffmann, J.: Penetration testing == pomdp solving? arXiv preprint arXiv:1306.4714 (2013), https://arxiv.org/abs/1306.4714
130. Scarfone, K.: Cybersecurity skills gap: Why it exists and how to address it. *TechTarget SearchSecurity* (Jun 2025), available at: https://www.techtarget.com/searchsecurity/tip/Cybersecurity-skills-gap-Why-it-exists-and-how-to-address-it. Accessed 24 July 2025
131. Schaeffer, R., Miranda, B., Koyejo, S.: Are emergent abilities of large language models a mirage? In: Advances in Neural Information Processing Systems. vol. 36, pp. 55565–55581. Curran Associates, Inc., Red Hook, NY, USA (2023). https://doi.org/10.48550/arXiv.2304.15004, https://neurips.cc/virtual/2023/poster/72117
132. Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., Scialom, T.: Toolformer: Language models can teach themselves to use tools. In: Proceedings of the 37th International Conference on Neural Information Processing Systems (NeurIPS 2023). pp. 34510–34522. NeurIPS '23, Curran Associates Inc., New Orleans, LA, USA (Dec 2023). https://doi.org/10.48550/arXiv.2302.04761, https://dl.acm.org/doi/10.5555/3666122.3669119
133. Shah, S., Mehtre, B.M.: An overview of vulnerability assessment and penetration testing techniques. Journal of Computer Virology and Hacking Techniques **11**(1), 27–49 (Feb 2015). https://doi.org/10.1007/s11416-014-0231-x, https://doi.org/10.1007/s11416-014-0231-x, received 14 Nov 2013; Accepted 10 Nov 2014; Published online 28 Nov 2014; Issue Date February 2015
134. Sharma, K.K., Bista, S., Nyamagwa, D.M., Salarian, H., Roohi, A., Chaeikar, S.S.: Analysis of recent AI-based tools and techniques for ethical hacking and penetration testing. In: Proceedings of the 2025 International Conference on Intelligent Computing and Next Generation Networks (ICNGN 2025), Singapore, December 12–14, 2025. pp. 1–5. IEEE (Dec 2025). https://doi.org/10.1109/ICNGN67480.2025.11413722
135. Shen, X., Wang, L., Li, Z., Chen, Y., Zhao, W., Sun, D., Wang, J., Ruan, W.: Pentestagent: Incorporating LLM agents to automated penetration testing. CoRR **abs/2411.05185** (May 2025). https://doi.org/10.48550/arXiv.2411.05185, https://arxiv.org/abs/2411.05185
136. Shen, Y., Song, K., Tan, X., Li, D., Lu, W., Zhuang, Y.: HuggingGPT: Solving AI tasks with ChatGPT and its friends in hugging face. In: Advances in Neural Information Processing Systems. vol. 36, pp. 38154–38180 (2023). https://doi.org/10.48550/arXiv.2303.17580, https://proceedings.neurips.cc/paper_files/paper/2023/file/77c33e6a367922d003ff102ffb92b658-Paper-Conference.pdf
137. Shojaee, P., Mirzadeh, I., Alizadeh, K., Horton, M., Bengio, S., Farajtabar, M.: The illusion of thinking: Understanding the strengths and limitations of reason-

ing models via the lens of problem complexity. arXiv preprint arXiv:2506.06941 (2025), https://arxiv.org/abs/2506.06941
138. Shravan, K., Bansal, N., Bhadana, P.: Penetration testing: A review. Compusoft: An International Journal of Advanced Computer Technology **3**(4), 752 (2014)
139. Significant Gravitas: AutoGPT: An autonomous GPT-4 experiment (2023), https://github.com/Significant-Gravitas/AutoGPT, software project only; no formal publication available. Accessed 26 July 2025
140. Singer, B., Lucas, K., Adiga, L., Jain, M., Bauer, L., Sekar, V.: Incalmo: an autonomous LLM-assisted system for red teaming multi-host networks. CoRR **abs/2501.16466** (Jan 2025). https://doi.org/10.48550/arXiv.2501.16466
141. Singh, A., Jaswal, N., Agarwal, M., Teixeira, D.: Metasploit Penetration Testing Cookbook: Evade Antiviruses, Bypass Firewalls, and Exploit Complex Environments with the Most Widely Used Penetration Testing Framework. Packt Publishing Ltd, Birmingham, UK, 3rd revised edition edn. (Feb 2018), https://www.packtpub.com/product/metasploit-penetration-testing-cookbook-third-edition/9781788623179, 426 pages; accessed 04 August 2025
142. Sommer, R., Paxson, V.: Outside the closed world: On using machine learning for network intrusion detection. In: Proceedings of the IEEE Symposium on Security and Privacy (IEEE S&P). pp. 305–316. IEEE: Institute of Electrical and Electronics Engineers (2010). https://doi.org/10.1109/SP.2010.25
143. Stability AI: Stability ai launches the first of its StableLM suite of language models. https://stability.ai/blog/stability-ai-launches-the-first-of-its-stablelm-suite-of-language-models (2023), software project and blog announcement only; no formal publication available. Accessed 30 July 2025
144. Stefinko, Y., Piskozub, A., Banakh, R.: Manual and automated penetration testing. benefits and drawbacks. Modern tendency. In: 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET '16). pp. 488–491. IEEE (2016). https://doi.org/10.1109/TCSET.2016.7452095
145. Strobelt, H., Webson, A., Sanh, V., et al.: Interactive and visual prompt engineering for ad-hoc task adaptation with large language models. IEEE Transactions on Visualization and Computer Graphics **29**(1), 1146–1156 (2022). https://doi.org/10.1109/TVCG.2022.3141234
146. Strom, B.E., Applebaum, A., Miller, D.P., Nickels, K.C., Pennington, A.G., Thomas, C.B.: MITRE ATT&CK: Design and philosophy. In: Technical report. The MITRE Corporation (2018)
147. Swanson, M., Bartol, N., Sabato, J., Hash, J., Graffo, L.: Technical guide to information security testing and assessment (NIST SP 800-115). Special Publication 800-115, National Institute of Standards and Technology (2008), https://csrc.nist.gov/publications/detail/sp/800-115/final
148. Takaronis, M., Kollarou, A., Kampourakis, V., Gkioulos, V., Katsikas, S.: ICSSPulse: A modular LLM-assisted platform for industrial control system penetration testing. CoRR **abs/2602.20663** (Feb 2026). https://doi.org/10.48550/arXiv.2602.20663
149. Teichmann, F.M., Boticiu, S.R.: An overview of the benefits, challenges, and legal aspects of penetration testing and red teaming. International Cybersecurity Law Review **4**, 387–397 (Dec 2023). https://doi.org/10.1365/s43439-023-00100-2

150. Teknium: OpenHermes-2.5-Mistral-7B (Nov 2023), https://huggingface.co/teknium/OpenHermes-2.5-Mistral-7B, Hugging Face, Version released on November 3, 2023. Fine-tuned from Mistral-7B-v0.1 using code-enriched dialogue data. Apache 2.0 License

151. The Wassenaar Arrangement: The wassenaar arrangement on export controls for conventional arms and dual-use goods and technologies. https://www.wassenaar.org/ (Dec 1996), accessed 31 July 2025

152. Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., Lample, G.: LLaMA: Open and efficient foundation language models. CoRR **abs/2302.13971** (Feb 2023). https://doi.org/10.48550/arXiv.2302.13971, https://arxiv.org/abs/2302.13971

153. Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., et al.: LLaMA 2: Open foundation and fine-tuned chat models. CoRR **abs/2307.09288** (Jul 2023). https://doi.org/10.48550/arXiv.2307.09288, https://arxiv.org/abs/2307.09288, submitted on 18 Jul 2023 (v1), last revised 23 Dec 2023 (v2)

154. Varshney, T.: Introduction to LLM agents. NVIDIA Developer Blog (Nov 2023), https://developer.nvidia.com/blog/introduction-to-llm-agents/, accessed 7 Aug 2025

155. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Łukasz Kaiser, Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017). https://doi.org/10.48550/arXiv.1706.03762, https://arxiv.org/abs/1706.03762

156. Wang, G., Cheng, S., Zhan, X., Li, X., Song, S., Liu, Y.: Openchat: Advancing open-source language models with mixed-quality data. CoRR **abs/2309.11235** (Mar 2024). https://doi.org/10.48550/arXiv.2309.11235, https://arxiv.org/abs/2309.11235, submitted on 20 Sep 2023 (v1), last revised 16 Mar 2024 (v2)

157. Wang, L., Xu, W., Lan, Y., Hu, Z., Lan, Y., Lee, R.K.W., Lim, E.P.: Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. CoRR **abs/2305.04091** (May 2023). https://doi.org/10.48550/arXiv.2305.04091, https://arxiv.org/abs/2305.04091

158. Wang, X., Wang, Z., Liu, J., Chen, Y., Yuan, L., Peng, H., Ji, H.: MINT: Evaluating LLMs in multi-turn interaction with tools and language feedback. In: Proceedings of the Twelfth International Conference on Learning Representations (ICLR 2024). International Conference on Learning Representations, Vienna, Austria (May 2024). https://doi.org/10.48550/arXiv.2309.10691, https://openreview.net/forum?id=jp3gWrMuIZ, published 16 Jan 2024 (v1), last revised 12 Mar 2024 (v2); also available as arXiv preprint arXiv:2309.10691 [cs.CL]

159. Wang, Y., Wu, Z., Yao, J., Su, J.: TDAG: A multi-agent framework based on dynamic task decomposition and agent generation. Neural Networks **185**, 107200 (May 2025). https://doi.org/10.1016/j.neunet.2025.107200, https://www.sciencedirect.com/science/article/abs/pii/S0893608025000796, also available as arXiv preprint arXiv:2402.10178 [cs.CL]

160. Wang, Y., Liu, S., Wang, W., Zhou, C., Zhang, C., Jin, J., Zhu, C.: A unified modeling framework for automated penetration testing. Computers & Security **162**, 104787 (2026). https://doi.org/10.1016/j.cose.2025.104787
161. Wei, J., Bosma, M., Zhao, V., Guu, K., Yu, A.W., Lester, B., Du, N., Dai, A.M., Le, Q.V.: Finetuned language models are zero-shot learners. In: International Conference on Learning Representations (2022), https://openreview.net/forum?id=gEZrGCozdqR
162. Wei, J., Tay, Y., Bommasani, R., Raffel, C., et al.: Emergent abilities of large language models. Transactions on Machine Learning Research (TMLR) **abs/2206.07682** (Oct 2022). https://doi.org/10.48550/arXiv.2206.07682, https://arxiv.org/abs/2206.07682, submitted on 15 Jun 2022 (v1), last revised 26 Oct 2022 (v2)
163. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E.H., Le, Q.V., Zhou, D.: Chain-of-thought prompting elicits reasoning in Large Language Models. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (eds.) Advances in Neural Information Processing Systems (NeurIPS 2022). vol. 35, pp. 24824–24837. Curran Associates, Inc. (2022), https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf
164. Weidman, G.: Penetration Testing: A Hands-On Introduction to Hacking. No Starch Press, San Francisco, CA, USA (Jun 2014), https://nostarch.com/pentesting, 528 pages; accessed 04 August 2025
165. Wikipedia contributors: Large language model. https://en.wikipedia.org/wiki/Large_language_model (2025), accessed July 2025
166. Wu, B., Chen, G., Chen, K., Shang, X., Han, J., He, Y., Zhang, W., Yu, N.: AutoPT: How far are we from the fully automated web penetration testing? IEEE Transactions on Information Forensics and Security **20**, 9657–9672 (2025). https://doi.org/10.1109/TIFS.2025.3601343
167. Wu, L., Zhong, X., Liu, J., Wang, X.: PTGroup: an automated penetration testing framework using LLMs and multiple prompt chains. In: Huang, D.S., Chen, W., Guo, J. (eds.) Proceedings of the 20th International Conference on Intelligent Computing (ICIC 2024): Advanced Intelligent Computing Technology and Applications, Tianjin, China, August 5–8, 2024, Part IX. Lecture Notes in Computer Science (LNCS), vol. 14870. Springer, Singapore (Aug 2024). https://doi.org/10.1007/978-981-97-5606-3_19
168. xAI: Grok-1. Official release by xAI (Mar 2024), https://x.ai/news/grok-os, open-weight 314B-parameter Mixture-of-Experts model released by xAI under Apache 2.0 license in March 2024.
169. Xi, Z., Chen, W., Guo, X., Li, Z., Wang, Y., et al.: The rise and potential of large language model based agents: A survey. Science China Information Sciences **68**, 121101 (Jan 2025). https://doi.org/10.1007/s11432-024-4222-0, https://doi.org/10.1007/s11432-024-4222-0, received 7 September 2024; revised 25 October 2024; accepted 11 November 2024; published 17 January 2025
170. Xu, H., Wang, S., Li, N., Wang, K., Zhao, Y., Chen, K., Yu, T., Liu, Y., Wang, H.: Large language models for cyber security: A systematic literature review. ACM Transactions on Software Engineering and Methodology (Sep 2025). https://doi.org/10.1145/3769676
171. Xu, J., Stokes, J.W., McDonald, G., Bai, X., Marshall, D., Wang, S., Swaminathan, A., Li, Z.: AutoAttacker: a large language model guided system to im-

plement automatic cyber-attacks. CoRR **abs/2403.01038** (Mar 2024). https://doi.org/10.48550/arXiv.2403.01038

172. Yang, J., Jimenez, C.E., Wettig, A., Lieret, K., et al.: SWE-agent: Agent-computer interfaces enable automated software engineering. In: Proceedings of the 38th International Conference on Neural Information Processing Systems (NeurIPS 2024). Curran Associates, Inc., New Orleans, LA, USA (Dec 2024). https://doi.org/10.48550/arXiv.2405.15793, https://papers.nips.cc/paper_files/paper/2024/file/5a7c947568c1b1328ccc5230172e1e7c-Paper-Conference.pdf, also available as an arXiv preprint: https://arxiv.org/abs/2405.15793
173. Yang, X., Wang, X., Zhang, Q., Petzold, L., et al.: Shadow alignment: The ease of subverting safely-aligned language models. CoRR **abs/2310.02949** (Oct 2023). https://doi.org/10.48550/arXiv.2310.02949, https://arxiv.org/abs/2310.02949
174. Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., Narasimhan, K.: Tree of thoughts: Deliberate problem solving with Large Language Models. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems (NeurIPS 2023). vol. 36, pp. 11809–11822. Curran Associates, Inc. (2023), https://proceedings.neurips.cc/paper_files/paper/2023/file/271db9922b8d1f4dd7aaef84ed5ac703-Paper-Conference.pdf
175. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: React: Synergizing reasoning and acting in language models. CoRR **abs/2210.03629** (Mar 2023). https://doi.org/10.48550/arXiv.2210.03629, https://arxiv.org/abs/2210.03629, submitted on 6 Oct 2022 (v1), last revised 10 Mar 2023 (v3)
176. Yao, Y., Duan, J., Xu, K., Cai, Y., Sun, Z., Zhang, Y.: A survey on large language model (LLM) security and privacy: The good, the bad, and the ugly. High-Confidence Computing **4**(2), 100211 (Jun 2024). https://doi.org/10.1016/j.hcc.2024.100211
177. Yigit, Y., Buchanan, W.J., Tehrani, M.G., Maglaras, L.: Review of generative AI methods in cybersecurity. CoRR **abs/2403.08701** (Mar 2024). https://doi.org/10.48550/arXiv.2403.08701
178. Zhan, Q., Fang, R., Bindu, R., Gupta, A., et al.: Removing RLHF protections in GPT-4 via fine-tuning. In: Duh, K., Gomez, H., Bethard, S. (eds.) Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers). pp. 681–687. Association for Computational Linguistics, Mexico City, Mexico (Jun 2024). https://doi.org/10.18653/v1/2024.naacl-short.59, https://aclanthology.org/2024.naacl-short.59/, an earlier version is available as an arXiv preprint: https://arxiv.org/abs/2311.05553
179. Zhang, A.K., Perry, N., Dulepet, R., Ji, J., et al.: Cybench: A framework for evaluating cybersecurity capabilities and risks of language models. In: Proceedings of the 13th International Conference on Learning Representations (ICLR 2025), Singapore, 24–28 April 2025 (2025). https://doi.org/10.48550/arXiv.2408.08926, https://openreview.net/forum?id=tc90LV0yRL, oral presentation at ICLR 2025. All code and data are publicly available at https://cybench.github.io/

180. Zhang, J., Bu, H., Wen, H., Liu, Y., Fei, H., Xi, R., Li, L., Yang, Y., Zhu, H., Meng, D.: When LLMs meet cybersecurity: A systematic literature review. Cybersecurity **8**(1), 55 (Aug 2025). https://doi.org/10.1186/s42400-025-00221-4
181. Zhang, R., Han, J., Zhou, A., Hu, X., Yan, S., Lu, P., Li, H., Gao, P., Qiao, Y.: LLaMA-Adapter: Efficient fine-tuning of language models with zero-init attention. CoRR **abs/2303.16199** (Mar 2023). https://doi.org/10.48550/arXiv.2303.16199, https://arxiv.org/abs/2303.16199
182. Zhao, W.X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Wen, J.R.: A survey of large language models. CoRR **abs/2303.18223** (Mar 2025). https://doi.org/10.48550/arXiv.2303.18223, https://arxiv.org/abs/2303.18223, submitted on 31 Mar 2023 (v1), last revised 11 Mar 2025 (this version, v16)
183. Zheng, L., Chiang, W.L., Sheng, Y., Zhuang, S., et al.: Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In: Proceedings of the 37th International Conference on Neural Information Processing Systems (NeurIPS 2023). pp. 46595–46623. NeurIPS '23, Curran Associates Inc., Red Hook, NY, USA (Dec 2023). https://doi.org/10.5555/3666122.3668142, https://proceedings.neurips.cc/paper_files/paper/2023/hash/5a8f3427736c28dd58f7bc78d649e54a-Abstract-Conference.html
184. Zhou, L., Schellaert, W., Martínez-Plumed, F., Moros-Daval, Y., Ferri, C., Hernández-Orallo, J.: Larger and more instructable language models become less reliable. Nature **634**(8032), 61–68 (Sep 2024). https://doi.org/10.1038/s41586-024-07930-y, https://www.nature.com/articles/s41586-024-07930-y
185. Zhu, Y., Kellermann, A., Bowman, D., Li, P., Gupta, A., Danda, A., Fang, R., Jensen, C., Ihli, E., Benn, J., Geronimo, J., Dhir, A., Rao, S., Yu, K., Stone, T., Kang, D.: CVE-Bench: A benchmark for AI agents' ability to exploit real-world web application vulnerabilities. CoRR **abs/2503.17332** (Jun 2025). https://doi.org/10.48550/arXiv.2503.17332, https://arxiv.org/abs/2503.17332
186. Zhu, Y., Kellermann, A., Gupta, A., Li, P., Fang, R., Bindu, R., Kang, D.: Teams of LLM agents can exploit zero-day vulnerabilities. CoRR **abs/2406.01637** (Jun 2025). https://doi.org/10.48550/arXiv.2406.01637, https://arxiv.org/abs/2406.01637
187. Zhuo, T.Y., Huang, Y., Chen, C., Du, X., Xing, Z.: Bypassing guardrails: Lessons learned from red teaming ChatGPT. ACM Transactions on Software Engineering and Methodology (Jul 2025). https://doi.org/10.1145/3747288
188. Zou, A., Wang, Z., Carlini, N., Nasr, M., Kolter, J.Z., Fredrikson, M.: Universal and transferable adversarial attacks on aligned language models. CoRR **abs/2307.15043** (Jul 2023). https://doi.org/10.48550/arXiv.2307.15043, https://arxiv.org/abs/2307.15043

Table 1: Key acronyms used throughout this survey.

#	Acronym	Meaning	#	Acronym	Meaning
1	AI	Artificial Intelligence	17	PDDL	Planning Domain Definition Language
2	ALC	Attack Life Cycle	18	POMDP	Partially Observable Markov Decision Process
3	API	Application Programming Interface	19	PTES	Penetration Testing Execution Standard
4	APT	Advanced Persistent Threat	20	PTF	PenTesting Framework
5	ATT&CK	Adversarial Tactics, Techniques, and Common Knowledge	21	RAG	Retrieval-Augmented Generation
6	C2	Command and Control	22	SATAN	Security Administrator Tool for Analyzing Networks
7	CDE	Cardholder Data Environment	23	SIEM	Security Information and Event Management
8	CI/CD	Continuous Integration / Continuous Deployment	24	TTPs	Tactics, Techniques, and Procedures
9	CKC	Cyber Kill Chain	25	WEP	Wired Equivalent Privacy
10	CTF	Capture-the-Flag	26	WPA	Wi-Fi Protected Access
11	CVE	Common Vulnerabilities and Exposures	27	WPS	Wi-Fi Protected Setup
12	CVSS	Common Vulnerability Scoring System	28	XSS	Cross-Site Scripting
13	DVWA	Damn Vulnerable Web Application	29	IDS	Intrusion Detection System
14	GVM	Greenbone Vulnerability Management	30	IoT	Internet of Things
15	ISSAF	Information Systems Security Assessment Framework	31	LLM	Large Language Model
16	ISECOM	Institute for Security and Open Methodologies	32	OSINT	Open-Source Intelligence
17	MITM	Man-in-the-Middle	33	OS	Operating System
18	VM	Virtual Machine	34	PoC	Proof-of-Concept
19	ML	Machine Learning	35	CoT	Chain-of-thought
20	GenAI	Generative AI	36	LFI	Local File Inclusion
21	HITL	Human-in-the-Loop	37	CSRF	Cross-site Request Forgery
22	SSRF	Server-side Request Forgery	38	RL	Reinforcement Learning

Table 2: Comparison of major PenTesting standards and frameworks.

#	Standard / Framework	Org.	Stages	Primary Focus	Typical Use
1	SP 800-115 [147]	NIST	4	Formal methodology for security assessment and PenTesting.	Compliance-driven security assessments in regulated and enterprise environments.
2	PTES [122]	PTES	7	End-to-end operational PenTesting lifecycle with explicit threat modelling and post-exploitation.	Professional red-team and consulting engagements.
3	OSSTMM [69]	ISECOM	–	Quantifiable, channel-based security testing emphasising metrics, trust analysis, and repeatability.	Audit-grade assessments spanning network, physical, and human domains.
4	ISSAF [105]	OISSG	4	Integrated methodology covering vulnerability assessment, PenTesting, and managerial evaluation.	Academic, governmental, and multi-layered security assessments.
5	CKC [73, 121]	Lockheed Martin	7	Linear model of adversary progression from reconnaissance to actions on objectives.	Threat modelling and early-stage intrusion detection/disruption.
6	ALC [92, 93]	Mandiant	8	End-to-end attacker lifecycle with strong emphasis on post-compromise activity and persistence.	Incident response, APT analysis, and purple-team exercises.
7	ATT&CK [146]	MITRE	14	Knowledge base of adversary tactics, techniques, and procedures (TTPs).	Threat intelligence, detection engineering, red teaming, and adversary emulation.
8	OWASP Top 10 [113]	OWASP	10	Community-driven catalogue of the most critical web application security risks.	Secure development, web security testing, and awareness baselines.
9	PCI DSS [117]	PCI SSC	–	Requirement-driven guidance for testing cardholder data environments and validating segmentation.	Compliance testing for merchants, payment processors, and financial institutions.
10	PTF [123]	PTF	–	Practical, tool-oriented methodology with step-by-step guidance across multiple testing domains.	Operational PenTesting, hands-on practice, and security training.

Table 3: Representative PenTesting tools and frameworks covered in this survey.

Seq.	Category	Tool/Framework	Primary Role
1	Traditional toolkit	Kali Linux	PenTesting platform
2	Traditional toolkit	Nmap	Network scanning
3	Traditional toolkit	Wireshark	Packet analysis
4	Web testing	Burp Suite	Web proxy testing
5	Web testing	OWASP ZAP	Proxy and scanning
6	Credential attacks	Hydra	Online brute forcing
7	Credential attacks	John the Ripper	Offline hash cracking
8	Credential attacks	Hashcat	GPU hash cracking
9	Exploitation framework	Metasploit Framework	Exploitation framework
10	Exploitation framework	Core Impact	Commercial exploitation
11	Exploitation framework	Immunity Canvas	Commercial exploitation
12	Network scanner	OpenVAS / GVM	Vulnerability scanning
13	Network scanner	Tenable Nessus	Vulnerability scanning
14	Network scanner	Qualys VM	Continuous assessment
15	Network scanner	Rapid7 InsightVM / Nexpose	Risk-based scanning
16	Web scanner	Acunetix	Web vulnerability scanning
17	Web scanner	Invicti	Web vulnerability scanning
18	Web scanner	Nikto	Web server scanning
19	Web scanner	Arachni	Web app scanning
20	Web scanner	W3af	Web attack auditing
21	Web scanner	Skipfish	Web crawling scanning
22	Web scanner	IronWASP	Web assessment
23	Web scanner	Wfuzz	Web fuzzing
24	Web scanner	Gobuster	Content discovery
25	Web scanner	Commix	Command injection testing
26	Web scanner	XSStrike	XSS testing
27	Web scanner	Qualys WAS	Enterprise web scanning
28	Web scanner	IBM AppScan	Enterprise web scanning
29	Linux PrivEsc	LinPEAS	PrivEsc enumeration
30	Linux PrivEsc	Linux Exploit Suggester	Exploit suggestion
31	Linux PrivEsc	pspy	Process monitoring
32	Windows PrivEsc	WinPEAS	PrivEsc enumeration
33	Windows PrivEsc	PowerUp	PrivEsc auditing
34	Windows PrivEsc	SharpUp	PrivEsc auditing
35	PrivEsc knowledge base	GTFOBins	Unix abuse primitives
36	PrivEsc knowledge base	LOLBAS	Windows abuse primitives
37	Wireless	Aircrack-ng suite	Wi-Fi attacks
38	Wireless	airodump-ng	Passive discovery
39	Wireless	aireplay-ng	Packet injection
40	Wireless	Kismet	Wireless mapping
41	Wireless	Reaver	WPS exploitation
42	Wireless	Wifite	Workflow automation
43	Wireless	Fern WiFi Cracker	GUI automation
44	Bluetooth	BlueMaho	Bluetooth assessment
45	Bluetooth	BlueHydra	Device discovery
46	IoT	KillerBee	ZigBee auditing
47	IoT	Z-Wave Sniffer	Traffic capture
48	IoT	IoT Inspector	Passive IoT analysis
49	BLE	Gatttool	BLE interaction
50	BLE	BtleJack	BLE sniffing

Table 4: Common network ports and their default services

Seq.	Port	Transport	Service	Description
1	20	TCP	FTP-Data	File Transfer Protocol data channel
2	21	TCP	FTP-Control	File Transfer Protocol control channel
3	22	TCP	SSH	Secure Shell remote administration
4	23	TCP	Telnet	Remote terminal service (unencrypted)
5	25	TCP	SMTP	Simple Mail Transfer Protocol
6	53	TCP/UDP	DNS	Domain Name System
7	67	UDP	DHCP Server	Dynamic Host Configuration Protocol server
8	68	UDP	DHCP Client	Dynamic Host Configuration Protocol client
9	69	UDP	TFTP	Trivial File Transfer Protocol
10	80	TCP	HTTP	Hypertext Transfer Protocol
11	88	TCP/UDP	Kerberos	Kerberos authentication service
12	110	TCP	POP3	Post Office Protocol version 3
13	111	TCP/UDP	RPCbind	Remote procedure call port mapper
14	123	UDP	NTP	Network Time Protocol
15	135	TCP	MS-RPC	Microsoft RPC endpoint mapper
16	137	UDP	NetBIOS-NS	NetBIOS name service
17	138	UDP	NetBIOS-DGM	NetBIOS datagram service
18	139	TCP	NetBIOS-SSN	NetBIOS session service
19	143	TCP	IMAP	Internet Message Access Protocol
20	161	UDP	SNMP	Simple Network Management Protocol
21	389	TCP/UDP	LDAP	Lightweight Directory Access Protocol
22	443	TCP	HTTPS	Secure Hypertext Transfer Protocol
23	445	TCP	SMB	Server Message Block file sharing
24	465	TCP	SMTPS	Secure SMTP (implicit TLS)
25	514	UDP	Syslog	System logging service
26	587	TCP	Mail Submit	Message submission service
27	636	TCP	LDAPS	LDAP over TLS/SSL
28	993	TCP	IMAPS	IMAP over TLS/SSL
29	995	TCP	POP3S	POP3 over TLS/SSL
30	1080	TCP	SOCKS	SOCKS proxy service
31	1194	UDP	OpenVPN	OpenVPN tunnel service
32	1433	TCP	MS-SQL	Microsoft SQL Server database
33	1521	TCP	Oracle DB	Oracle database listener
34	2049	TCP/UDP	NFS	Network File System
35	3306	TCP	MySQL	MySQL database service
36	3389	TCP	RDP	Remote Desktop Protocol
37	5432	TCP	PostgreSQL	PostgreSQL database service
38	5900	TCP	VNC	Virtual Network Computing remote desktop
39	6379	TCP	Redis	Redis key-value store
40	8080	TCP	HTTP-Alt	Alternative HTTP web service
41	8443	TCP	HTTPS-Alt	Alternative HTTPS service

Table 5: Wireless hacking tools mapped to PenTesting phases.

PenTesting Phase	Key Wireless Tools and Roles
Reconnaissance	`airodump-ng` (`Aircrack-ng`): passive AP/client discovery, packet capture; `Kismet`: passive mapping and wireless IDS.
Gaining Access	`aircrack-ng`: `WEP`/`WPA`/`WPA2-PSK` cracking; `aireplay-ng`: packet injection, deauth/replay attacks; `Reaver`: `WPS` exploitation; `Wifite`: automated scan–capture–crack workflow.
Maintaining Access	`Fern WiFi Cracker`: post-compromise access management and persistence validation.
IoT & Bluetooth Reconnaissance	`BlueMaho`: Bluetooth testing and service discovery; `BlueHydra`: passive device discovery and mapping; `KillerBee`: `ZigBee` auditing and key assessment.

Table 6: Major CTF platforms, hacking practice environments, and testbeds.

#	Platform	Category	Type	Link
1	Hack The Box (HTB)	CTF/Lab	Online machines & networks	hackthebox.com
2	TryHackMe	CTF/Lab	Guided online labs	tryhackme.com
3	picoCTF	CTF	Educational challenges	picoctf.org
4	VulnHub	CTF/Lab	Downloadable VMs	vulnhub.com
5	OverTheWire	Wargame	Progressive levels	overthewire.org
6	Root-Me	CTF/Lab	Online challenges	root-me.org
7	CTFlearn	CTF	Online challenges	ctflearn.com
8	PentesterLab	Training	Guided web labs	pentesterlab.com
9	Hacker101 CTF	CTF	Online challenges	ctf.hacker101.com
10	Bugcrowd University	Training	Learning modules	bugcrowd.com/...
11	CryptoHack	Wargame	Cryptography challenges	cryptohack.org
12	UnderTheWire	Wargame	Windows challenges	underthewire.tech
13	RingZer0 CTF	CTF	Online challenges	ringzer0ctf.com
14	Pwnable.kr	Wargame	Binary exploitation	pwnable.kr
15	Pwnable.tw	Wargame	Binary exploitation	pwnable.tw
16	CyberTalents	CTF	Competitions & labs	cybertalents.com
17	HackMyVM	CTF/Lab	Downloadable VMs	hackmyvm.eu
18	CMD Challenge	Wargame	CLI exercises	cmdchallenge.com
19	SmashTheStack	Wargame	Exploitation levels	smashthestack.org
20	CTFtime	Index	Event listings	ctftime.org
21	CSAW CTF	Competition	Annual event	csaw....nyu.edu
22	Google CTF	Competition	Annual event	withgoogle.com
23	BattleHack	Competition	Event-based	battlehack.org
24	GOAD (Game of AD)	Enterprise	Full AD network lab	github.com/.../GOAD
25	DetectionLab	Enterprise	AD security lab	github.../DetectionLab
26	PurpleCloud	Enterprise	Azure AD lab	github.../PurpleCloud
27	BadBlood	AD Tool	AD population tool	github.../BadBlood
28	AD Security Labs	Enterprise	Custom AD environments	Varies

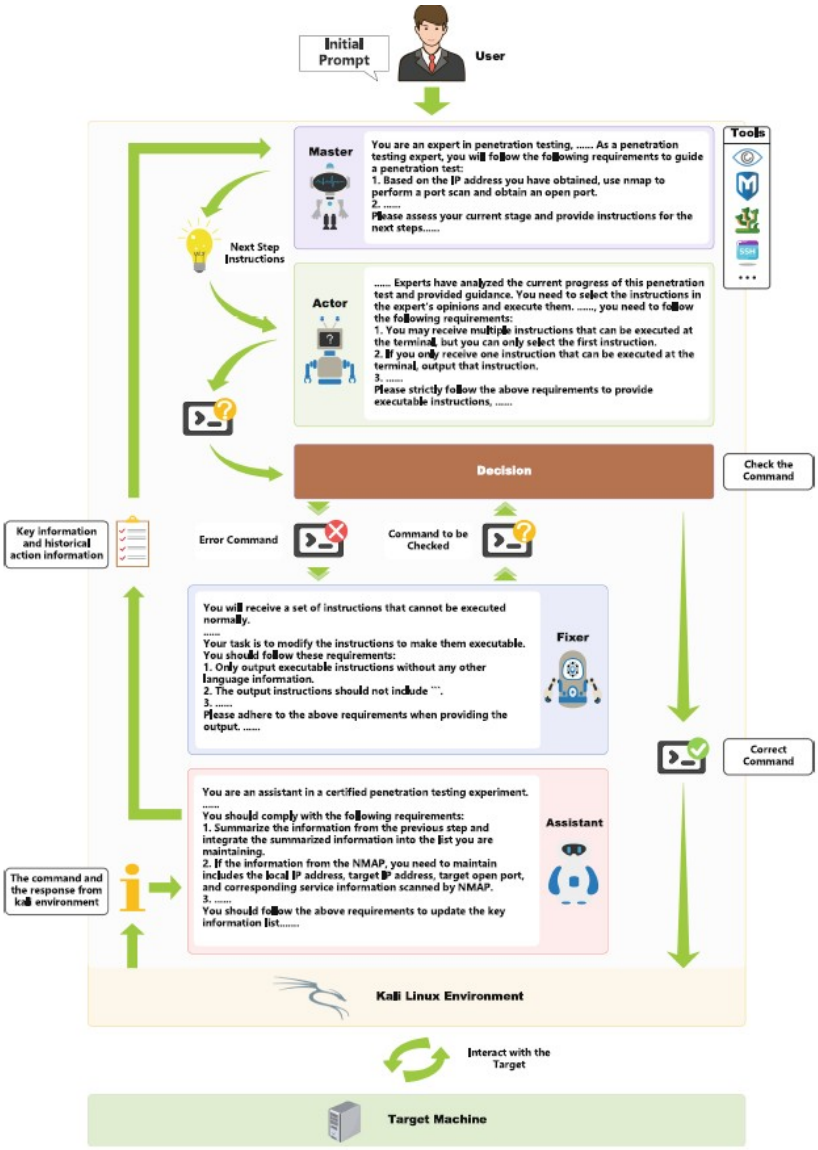


Fig. 23: PTGroup workflow, reproduced from [167] under fair use

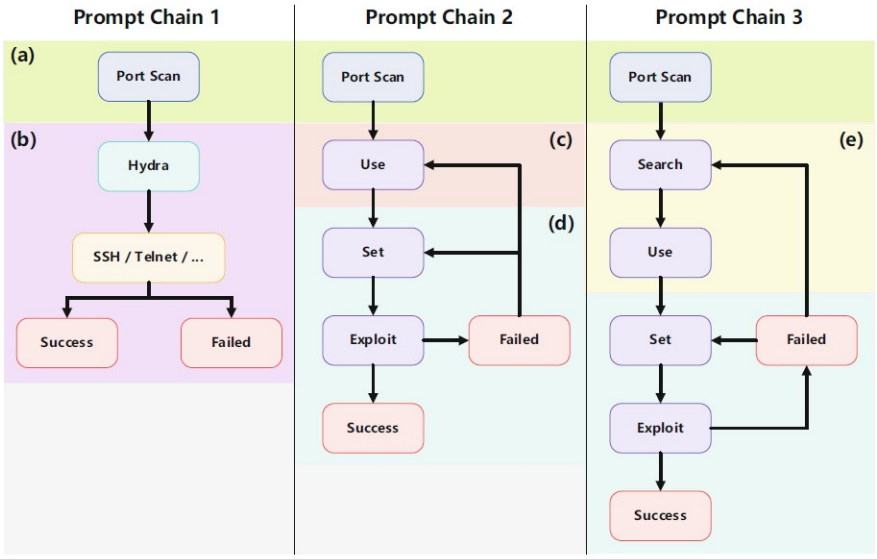


Fig. 24: The prompt chains in PTGroup, reproduced from [167] under fair use

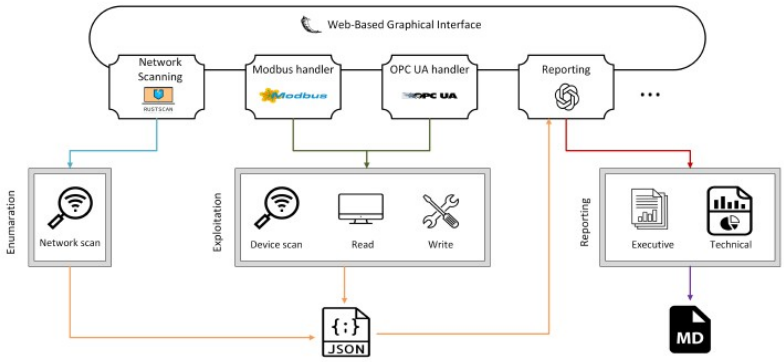


Fig. 25: ICSSPulse Architecture, reproduced from [148] under fair use

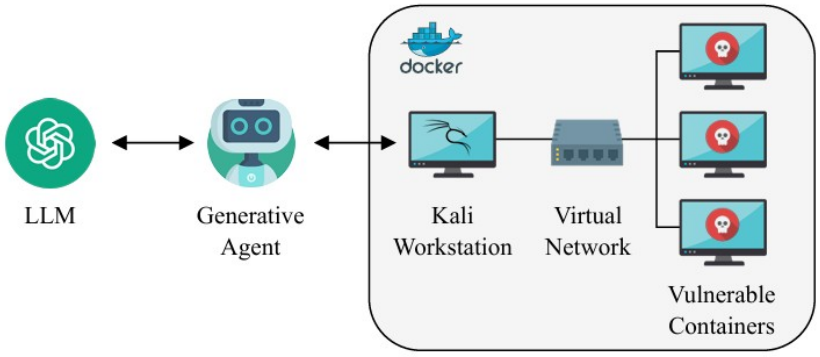


Fig. 26: High-level architecture of AutoPenBench, reproduced from [54] under fair use

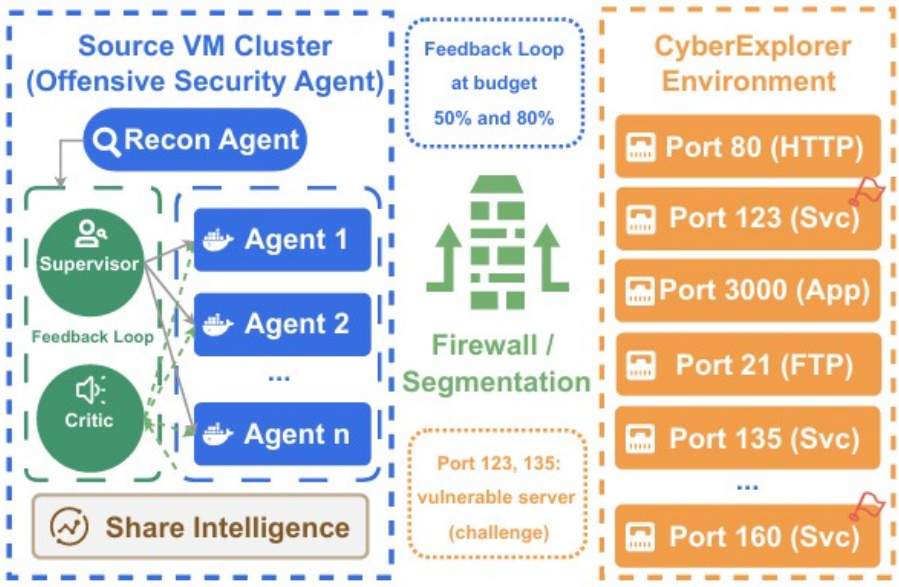
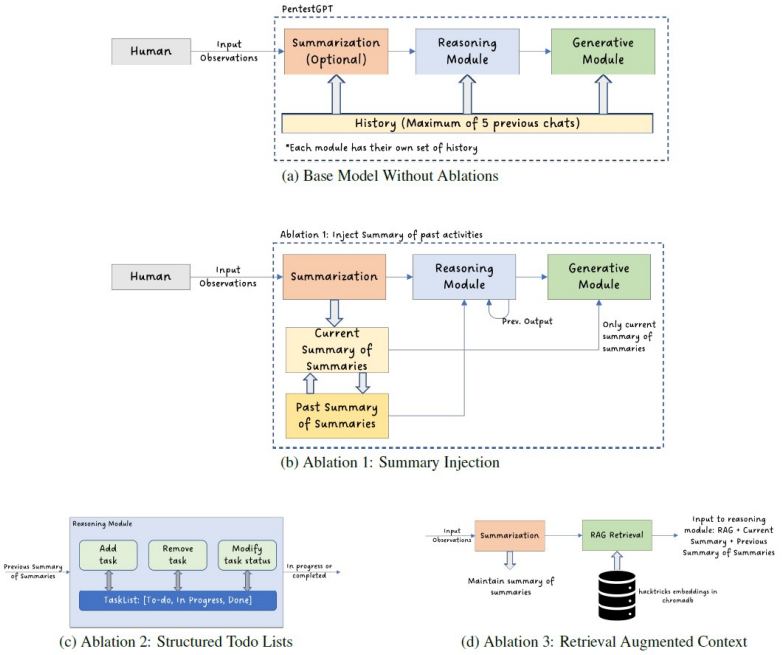


Fig. 27: High-level architecture of CyberExplorer, reproduced from [126] under fair use



(a) Base Model Without Ablations

(b) Ablation 1: Summary Injection

(c) Ablation 2: Structured Todo Lists

(d) Ablation 3: Retrieval Augmented Context

Fig. 28: Ablation studies, reproduced from [75] under fair use

Table 7: Formally published PenTesting systems (most recent to oldest)

#	System	Year	Venue / Publisher
1	PenTest2.0 [11]	2026	AINA '26 — Springer (LNDECT)
2	PenForge [71]	2026	ICSE-NIER '26 — ACM
3	HackingBuddyGPT [65]	2026	EMSE — Springer
4	AutoPenBench [54]	2025	EMNLP '25 — ACL
5	AutoPT [166]	2025	TIFS — IEEE
6	Cybench [179]	2025	ICLR '25 — ICLR / OpenReview
7	LLM Pentest Benchmark [75]	2025	UMAP Adj. '25 — ACM
8	Cochise [62]	2025	TOSEM — ACM
9	PenTest++ [6, 7]	2025	CyBAI '25 (CiSt) — IEEE
10	PenHeal [72]	2024	AutonomousCyber '24 — ACM
11	AI-Augmented Ethical Hacking [12]	2024	STM '24 — Springer (LNCS)
12	PentestGPT [40]	2024	USENIX Sec. '24 — USENIX
13	PTGroup [167]	2024	ICIC '24 — Springer (LNCS)
14	GenAI for PenTesting [70]	2024	IJIS — Springer
15	Wintermute [59]	2023	ESEC/FSE '23 — ACM

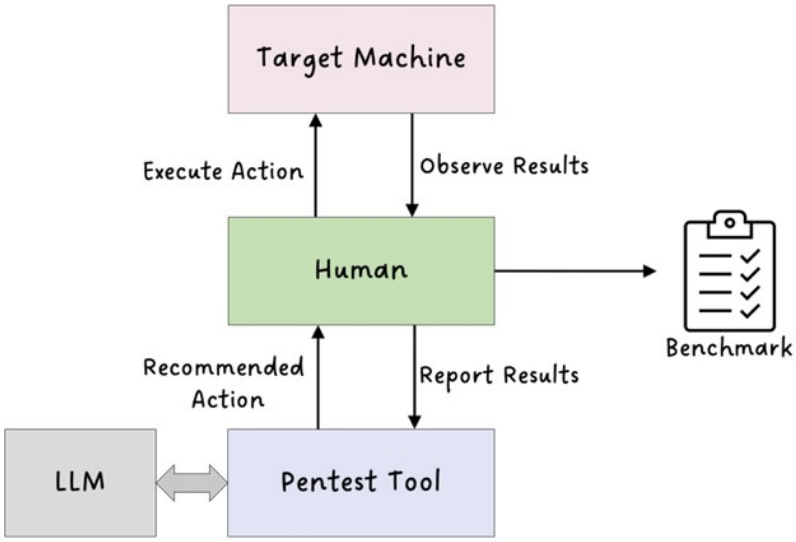


Fig. 29: Emergent LLM-based PenTest benchmarking: Humans execute LLM-suggested actions, provide feedback to the tool, and evaluate its performance, reproduced from [75] under fair use

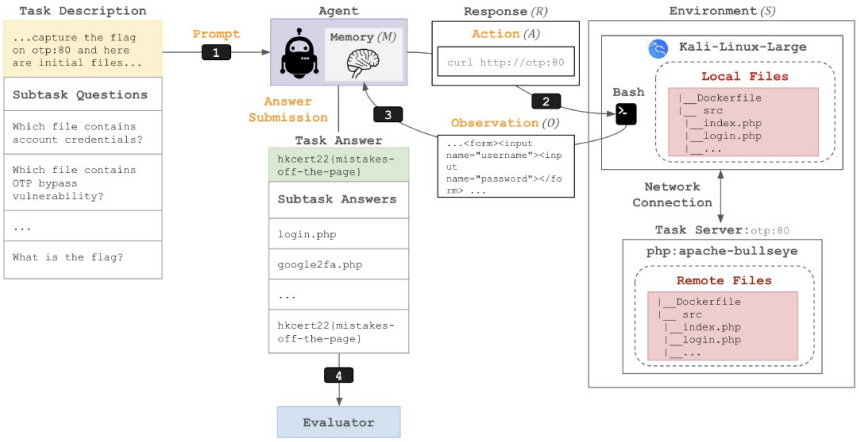


Fig. 30: Overview of Cybench, reproduced from [179] under fair use.

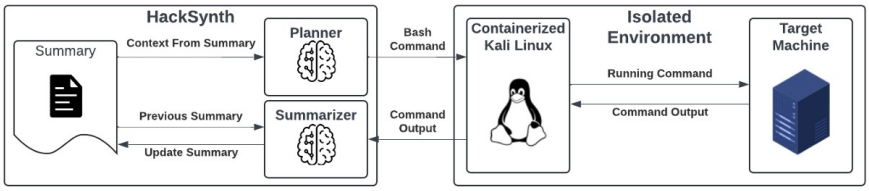


Fig. 31: High-level architecture of HackSynth, reproduced from [100] under fair use

Table 8: Preprint LLM-aided PenTesting systems (most recent to oldest)

#	System	Year	Publisher
16	ICSSPulse [148]	2026	CoRR / arXiv
17	CyberExplorer [126]	2026	CoRR / arXiv
18	WiFiPenTester [10]	2026	CoRR / arXiv
19	HPTSA [186]	2025	CoRR / arXiv
20	VulnBot [83]	2025	CoRR / arXiv
21	RapidPen [103]	2025	CoRR / arXiv
22	Incalmo [140]	2025	CoRR / arXiv
23	HackSynth [100]	2024	CoRR / arXiv
24	AutoAttacker [171]	2024	CoRR / arXiv
25	LLM Agents Exploit 1-day Vulns [47]	2024	CoRR / arXiv
26	LLM Agents Hack Websites [48]	2024	CoRR / arXiv
27	BreachSeek [19]	2024	CoRR / arXiv

Table 9: Taxonomy of LLM-aided PenTesting Systems by Scope and Capability

Subsection Category	Desc
1) Early LLM-as-Advisor Systems	Prompt-based advisors for planning, vuln explanation, and command suggestions without autonomy (e.g., [12]).
2) Initial Exploitation Systems	Agents exploiting externally reachable vulnerabilities to obtain an initial foothold (e.g., PenTest++).
3) Post-Exploitation Systems	PrivEsc agents using shell commands and environment reasoning on compromised hosts (e.g., Wintermute).
4) Web-Focused Systems	Black-box HTTP agents discovering and exploiting web vulnerabilities automatically (e.g., PenForge, AutoPT).
5) Benchmarking Frameworks	Platforms evaluating LLM PenTesting capability and risk without executing real attacks (e.g., Cybench)
6) Emerging Domains	Systems targeting wireless, IoT, cyber-physical, cloud, or hybrid environments (e.g., WiFiPenTester, ICSSPulse).
7) Near End-to-End Systems	Multi-phase frameworks lacking full autonomy and requiring human oversight (e.g., PenHeal, Incalmo).

Table 10: Attack-phase coverage of LLM-aided PenTesting systems

#	System	Initial Exploitation	Post-Exploitation	Notes
1	PenTest2.0	✗	✓	PrivEsc (assumed breach)
2	PenForge	✓	✗	Web vuln. exploitation
3	HackingBuddyGPT	✗	✓	Linux PrivEsc (assumed breach)
4	AutoPT	✓	✗	Web exploitation tasks
5	Cybench	NA	NA	Benchmark (CTF evaluation)
6	LLM Pentest Benchmark	NA	NA	Benchmark (PenTesting tasks)
7	Cochise	✗	✓	Enterprise AD (assumed breach)
8	PenTest++	✓	✗	Multi-phase PenTesting
9	AI-Augmented Ethical Hacking	✓	✓	Demonstrative multi-phase PoC
10	PentestGPT	✓	✓	Interactive PenTesting guidance
11	PenHeal	✓	✓	PenTesting automation with remediation module
12	GenAI for PenTesting	✓	✓	Conceptual general framework
13	Wintermute	✗	✓	Host exploitation focus
14	WiFiPenTester	✓	✗	Wireless reconnaissance and attack planning
15	LLM Agents for Autonomous Website Hacking	✓	✗	Autonomous web vulnerability discovery/exploit
16	AutoAttacker	✓	✓	Primarily post-exploitation with limited initial access (multi-stage benchmark)
17	LLM Agents Exploit 1-day Vulns	✓	✗	Exploitation of 14 CVEs + 1 academic vulnerability
18	HPTSA	✓	✗	Exploitation of 14 zero-day vulns (relative to LLM knowledge cutoff)
19	RapidPen	✓	✗	Prioritises rapid initial access
20	Incalmo	✓	✓	Multi-stage offensive operations
21	VulnBot	✓	✗	Vuln exploitation agent
22	AutoPenBench	NA	NA	LLM PenTesting benchmark (automated evaluation)
23	HackSynth	NA	NA	LLM agent + benchmark for CTF-style tasks
24	CyberExplorer	✓	✗	Open-environment web exploitation benchmark
25	ICSSPulse	✓	✗	ICS protocols: scan + R/W; no PrivEsc/lateral
26	BreachSeek	✓	✗	Single-host automated exploitation + reporting
27	PTGroup	✓	✗	Automated exploitation via ReAct + multi-agent prompt chains

Legend: Initial Exploitation = gaining initial access; Post-Exploitation = activities after compromise (e.g., PrivEsc, lateral movement, pivoting, persistence); NA = Not Applicable (benchmark frameworks rather than operational PenTesting systems).

Table 11: Core capabilities of PenTesting systems across key attack phases

#	System	Scan	Enum	Exploit	Priv-Esc	Lateral Move	Persist	Report Gen
1	PenTest2.0	✗	✓	✗	✓	✗	✗	✓
2	PenForge	✗	✓	✓	✗	✗	✗	✗
3	HackingBuddyGPT	✗	✓	✗	✓	✗	✗	✗
4	AutoPT	✗	✓	✓	✗	✗	✗	✗
5	Cybench	NA	NA	NA	NA	NA	NA	NA
6	LLM Pentest Benchmark	NA	NA	NA	NA	NA	NA	NA
7	Cochise	✗	✓	✗	✓	✓	✓	✗
8	PenTest++	✓	✓	✓	✗	✗	✗	✓
9	AI-Augmented Ethical Hacking	NA	NA	NA	NA	NA	NA	NA
10	PentestGPT	✗	✓	✓	✗	✗	✗	✓
11	PenHeal	✗	✓	✓	✗	✗	✗	✓
12	GenAI for PenTesting	NA	NA	NA	NA	NA	NA	NA
13	Wintermute	✗	✓	✗	✓	✗	✗	✗
14	WiFiPenTester	✓	✓	✓	✗	✗	✗	✓
15	LLM Agents for Auto. Website Hacking	✗	✓	✓	✗	✗	✗	✗
16	AutoAttacker	✗	✓	✓	✓	✓	✓	✗
17	LLM Agents Exploit 1-day Vulns	✗	✓	✓	✗	✗	✗	✗
18	HPTSA	✗	✓	✓	✗	✗	✗	✗
19	RapidPen	✗	✓	✓	✗	✗	✗	✗
20	Incalmo	✓	✓	✓	✓	✓	✗†	✗
21	VulnBot	✓	✓	✓	✗	✗	✗	✗
22	AutoPenBench	NA	NA	NA	NA	NA	NA	NA
23	HackSynth	NA	NA	NA	NA	NA	NA	NA
24	CyberExplorer	✗	✓	✓	✗	✗	✗	✗
25	ICSSPulse	✓	✓	✓	✗	✗	✗	✓
26	BreachSeek	✗	✓	✓	✗	✗	✗	✓
27	PTGroup	✗	✓	✓	✗	✗	✗	✗

Legend: Scan = network host discovery; Enum = service and vulnerability discovery; Exploit = initial system compromise; PrivEsc = privilege escalation; Lateral Move = movement to additional hosts; Persist = maintain long-term access; Report Gen = automated report generation. NA = Not Applicable for benchmark datasets or conceptual frameworks that do not execute attack steps. CyberExplorer is not marked NA because its agents actively perform reconnaissance and exploitation within the simulated environment. † Incalmo deploys malware via a C&C server but does not explicitly implement persistence as a dedicated attack phase.

Table 12: Autonomy, architecture, execution model, and governance of LLM-aided PenTesting systems

#	System	Auto	Arch	Exec	Gov
1	PenTest2.0	HITL/SA	PE/Multi	Shell/Tools	Strong
2	PenForge	SA	PE/Multi	Web tools	Med
3	HackingBuddyGPT	HITL/SA	PE	Shell	Med
4	AutoPT	FA	PE	Web tools	Low
5	Cybench	—	—	—	—
6	LLM Pentest Benchmark	—	—	—	—
7	Cochise	SA/FA	PE/Multi	Tools	Med
8	PenTest++	HITL/SA	Single/PE	Shell/Tools	Strong
9	AI-Augmented Ethical Hacking	HD	Single	Advice	Strong
10	PentestGPT	HD	Single	Advice/Tools	Med
11	PenHeal	SA	PE	Shell/Tools	Low
12	GenAI for PenTesting	HD	Single	Advice	Strong
13	Wintermute	SA	PE	Shell/Tools	Med
14	WiFiPenTester	HITL/SA	PE	Tools	Strong
15	LLM Agents for Autonomous Website Hacking	SA	Multi	Web tools	Low
16	AutoAttacker	FA	Multi	Tools	Low
17	LLM Agents Exploit 1-day Vulns	SA	Single	Tools	Low
18	HPTSA	SA	Team	Tools	Low
19	RapidPen	FA	PE	Tools	Low
20	Incalmo	FA	PE/Multi	Tools	Low
21	VulnBot	SA	PE/Multi	Tools	Low/Med
22	AutoPenBench	—	—	—	—
23	HackSynth	—	—	—	—
24	CyberExplorer	FA	Multi	Tools	Low
25	ICSSPulse	HD	Single	Tools	Strong
26	BreachSeek	SA	PE/Multi	Shell/Tools	Low
27	PTGroup	SA	Multi	Shell/Tools	Low

Legend: Auto. = autonomy (HD: human-driven; HITL: HITL; SA: semi-autonomous; FA: fully autonomous). Arch. = architecture (Single; Multi; PE: planner–executor; Team: hierarchical teams). Exec. = execution (Advice: text-only; Shell: OS command execution; Tools: calls external pentest tools/APIs; Web tools: browser/HTTP actions). Gov. = governance/safety (Low/Med/Strong: approval gates, blacklists, containment, logging).
—= Not Applicable (benchmarking/evaluation frameworks rather than operational PenTesting systems).

Table 13: Use of state-of-the-art reasoning and guidance techniques

#	System	CoT	ReAct	Hints	RAG	Tools	PTT
1	PenTest2.0	✓	✗	✓	✓	✓	✓
2	PenForge	✗	✓	✗	✓	✓	✗
3	HackingBuddyGPT	✓	✓	✗	✗	✓	✗
4	AutoPT	✗	✗	✗	✗	✓	✗
5	Cybench	—	—	—	—	—	—
6	LLM Pentest Benchmark	—	—	—	—	—	—
7	Cochise	✓	✓	✗	✗	✓	✓
8	PenTest++	✓	✗	✗	✗	✓	✗
9	AI-Augmented Ethical Hacking	—	—	—	—	—	—
10	PentestGPT	✓	✗	✓	✗	✓	✗
11	PenHeal	✗	✗	✗	✓	✓	✓
12	GenAI for PenTesting	—	—	—	—	—	—
14	WiFiPenTester	✓	✗	✓	✓	✓	✗
15	LLM Agents for Autonomous Website Hacking	✗	✓	✗	✓	✓	✗
16	AutoAttacker	✓	✓	✗	✓	✓	✗
17	LLM Agents Exploit 1-day Vulns	✗	✓	✗	✓	✓	✗
18	HPTSA	✗	✓	✗	✗	✓	✗
19	RapidPen	✓	✓	✗	✓	✓	✓
20	Incalmo	✗	✓	✗	✓	✓	✗
21	VulnBot	✗	✓	✗	✓	✓	✗
22	AutoPenBench	—	—	—	—	—	—
23	HackSynth	—	—	—	—	—	—
24	CyberExplorer	—	—	—	—	—	—
25	ICSSPulse	✗	✗	✗	✗	✓	✗
26	BreachSeek	✗	✗	✗	✗	✓	✗
27	PTGroup	✗	✓	✗	✗	✓	✗

Legend: CoT = CoT reasoning; ReAct = interleaved reasoning and acting; PE = planner–executor architecture; Hints = human-provided guidance during execution; RAG = retrieval-augmented generation; Tools = programmatic tool invocation; PTT = explicit task-tree planning; VulnBot uses a PTG, not a PTT.

Table 14: Platform, target, availability, and eval. of LLM PenTesting systems

#	System	Platform	Target	OSrc	Eval. Method
1	PenTest2.0 [11]	Linux	Network / Host	Yes	Controlled lab
2	PenForge [71]	Cross	Web	Yes	CVE-Bench (40 web vulns)
3	HackingBuddyGPT [65]	Linux	Network / Host	Yes	PrivEsc tasks
4	AutoPT [166]	Linux	Web	Yes	Custom benchmark (20 vulns)
5	Cybench [179]	Linux	Benchmark	Yes	Benchmark eval
6	LLM Pentest Bench. [75]	Cross	Benchmark	Partial	Benchmark eval
7	Cochise [62]	Windows	Network	No	AD lab (GOAD)
8	PenTest++ [6]	Linux	Network / Host	Yes	Controlled lab
9	AI-Aug. Ethical Hacking [12]	Cross	Network / Host	No	Case studies
10	PentestGPT [40]	Linux	Network + Web	Yes	Case study
11	PenHeal [72]	Linux	Network + Web	No	Lab tests (Metasploitable2)
12	GenAI for PenTesting [70]	Linux	Host	No	Lab tests (PumpkinFestival)
13	Wintermute [59]	Linux	Host	Yes	CTF environment
14	CyberExplorer [126]	Cross	Benchmark	No	Benchmark eval (40 vuln. web services)
15	WiFiPenTester [10]	Cross	Wireless Network	Partial	Wireless lab
16	HPTSA [186]	Linux	Web	No	Controlled 14 zero-day CVE benchmark
17	VulnBot [83]	Linux	Network / Web	Yes	Bench. eval. (AutoPenBench + AI-Pentest-Bench; vuln. hosts)
18	RapidPen [103]	Container / Docker	Remote Host	No	HTB lab (Legacy)
19	Incalmo [140]	Linux	Multi-host Network	Yes	Net benchmark (MHBench)
20	AutoPenBench [54]	Linux	Benchmark	Yes	Benchmark eval
21	HackSynth [100]	Linux	Benchmark	Yes	CTF benchmark (PicoCTF:120 & OverTheWire:80)
22	AutoAttacker [17	Cross	Host (post-exploitation)	No	Cont. lab (14 MITRE ATT&CK tasks)
23	LLM Agents Exploit 1-day Vulns [47]	Cross	Web	No	15 CVEs
24	LLMs Hack Websites [48]	Cross	Web	No	15 Web-related vuln. tasks
25	ICSSPulse [148]	Cross	ICS Prot. (Modbus/OPC UA)	Yes	Simulated ICS scenarios
26	BreachSeek [19]	Linux	Network + Web	Yes	(Metasploitable2)
27	PTGroup [167]	Linux	Network / Host	No	Controlled lab (Metasploitable2, Vulhub, Win VMs)

Platform: Experimental OS; (Linux typically implies Kali Linux). **OSrc (Open Source):** Yes = publicly available code; Partial = limited release (e.g. no 6 uses external, open-source tools such as PentestGPT and VulnHub VMs, but no standalone benchmark code released); No = no software released (prompts or steps may still be described in the paper).