

PenGPT

*LLM-Guided Red-Team Automation
With Persistent Shell Execution*

WRITTEN BY
Arabella Moore
Xing Liu, Ph.D
Narasimha K. Shashidhar Ph.D
Bing Zhou, Ph.D

2026



**INSTITUTE FOR
HOMELAND SECURITY**
SAM HOUSTON STATE UNIVERSITY



PenGPT: LLM-Guided Red-Team Automation With Persistent Shell Execution

Arabella Moore, Xing Liu, Narasimha K. Shashidhar, Bing Zhou

Department of Computer Science, Sam Houston State University
Huntsville, TX, USA

ABSTRACT

Red-team workflows are fragmented across many tools and require operators to translate high-level goals into system-specific command sequences. Large language models (LLMs) provide a promising interface for planning and command generation, but practical use still requires execution control, interactive tool support, traceability, and safety checks. This paper presents PenGPT, a prototype framework for LLM-guided red-team automation. PenGPT converts a user task into structured command actions, validates proposed steps, executes commands in persistent Linux or Windows shell sessions, supports interactive tools such as Netcat and Metasploit-style workflows, and augments prompts with a Neo4j-backed cybersecurity knowledge graph. We describe the system architecture and report preliminary case studies on password cracking, initial access, and exploitation-oriented workflows in an isolated virtualized lab. The results show that PenGPT can complete representative multi-step tasks while preserving operator oversight and execution traceability. We position PenGPT as a prototype and discuss current limitations, including the lack of large-scale quantitative evaluation and the need for stronger safety enforcement beyond pattern-based filtering.

Index Terms—large language models, red teaming, penetration testing, autonomous agents, command generation, cybersecurity automation

TABLE OF CONTENTS

ABSTRACT	2
I. INTRODUCTION	2
II. RELATED WORK	3
III. PenGPT DESIGN	4
A. Architecture Overview	4
B. Structured Action Generation	4
C. Persistent Execution and Interactive Tool Support	5
D. Knowledge-Graph Augmentation	5
E. Execution Loop	6
F. Validation, Human Oversight, and Logging	6
IV. PROTOTYPE EVALUATION	7
A. Experimental Setting	7
B. Case Study 1: Password Cracking Workflow	7
C. Case Study 2: Initial Access Through Interactive Shell Use	7
D. Observed Strengths and Failure Modes	7
E. Case Study 3: Tool-Oriented Exploitation Flow	8
V. DESIGN IMPLICATIONS FOR LLM-ASSISTED RED TEAMING	9
VI. LIMITATIONS AND ETHICAL CONSIDERATIONS	10
VII. CONCLUSION	10
AUTHOR BIOGRAPHIES	11
REFERENCES	12

I. INTRODUCTION

Red teaming and penetration testing remain labor-intensive because operators must decompose high-level goals into tool-specific commands, interpret outputs, and adapt their workflow to the target environment. Even common engagements may require several utilities, such as network scanners, exploitation frameworks, and credential recovery tools, each with different interfaces and assumptions. This fragmentation creates a barrier for new practitioners and slows down experienced analysts when tasks require repeated context switching across tools and operating systems.

Recent large language models (LLMs) have shown strong capability in code and command generation, structured reasoning, and task decomposition. These properties make LLMs attractive as planning interfaces for cybersecurity workflows. However, command generation alone is not enough for practical red-team automation. An operational system must maintain execution state, support interactive command-line tools, validate unsafe or incompatible commands, and preserve logs for later review.

This paper presents PenGPT, a prototype framework for LLM-guided red-team automation. PenGPT accepts a high-level user objective, retrieves task-relevant context from a cybersecurity knowledge graph, prompts an LLM to produce structured command actions, validates those actions, and executes them in persistent shell sessions. The system supports both ordinary shell commands and interactive sessions, while keeping human oversight in the loop when manual input or authorization is needed.

Rather than claiming a fully validated autonomous penetration testing system, we frame PenGPT as a prototype intended to explore how LLMs can coordinate red-team workflows under operator oversight. In this version, we emphasize the architectural elements that matter most in practice: persistent execution state, structured command generation, support for interactive tools, lightweight safety checks, knowledge-grounded prompting, and detailed session logging.

II. RELATED WORK

Existing LLM-assisted command and agent systems span at least three relevant categories. The first category focuses on command translation. ShellGPT maps natural-language requests to shell commands and is effective for single-command assistance, but it does not maintain execution state or perform end-to-end task execution. The second category focuses on penetration-testing planning agents. PentestGPT decomposes a task into subgoals and maintains a structured reasoning loop, but it still relies on human oversight for execution and does not emphasize persistent shell management or built-in safety validation. The third category consists of general-purpose autonomous agents such as Auto-GPT, which demonstrate plan-and-execute loops but are not grounded in cybersecurity-specific tool knowledge and may drift from task objectives.

PenGPT combines several capabilities that are usually separated across these systems: cybersecurity-oriented planning, persistent shell execution, support for interactive tools, lightweight command validation, and detailed trace logging. Table 1 summarizes the qualitative differences.

TABLE I
QUALITATIVE COMPARISON OF RELATED LLM-DRIVEN TOOLS

Feature	ShellGPT	PentestGPT	Auto-GPT	PenGPT
Task planning	Limited	Multi-step	General loop	Cyber-task loop
Persistent shell	No	No	Limited	Yes
Interactive tools	No	Limited	Limited	Yes
Safety validation	No	No	No	Pattern-based
Domain grounding	Generic	Pentest context	Generic	Cyber knowledge graph
Trace logging	Minimal	Structured state	Memory/file	Detailed logs

III. PENGPT DESIGN

A. Architecture Overview

PenGPT follows a plan-and-execute design. A user submits a task such as password cracking or initial access. The system retrieves relevant context from a Neo4j-backed knowledge graph, builds a system prompt, and queries an LLM to generate either ordinary shell commands or an interactive-session specification. Each proposed action passes through a command validator before execution. Standard commands are executed in a persistent Linux or Windows shell, while interactive commands are delegated to a dedicated tool handler. Command outputs and return information are then fed back into the loop so that the model can continue, revise, or terminate the task.

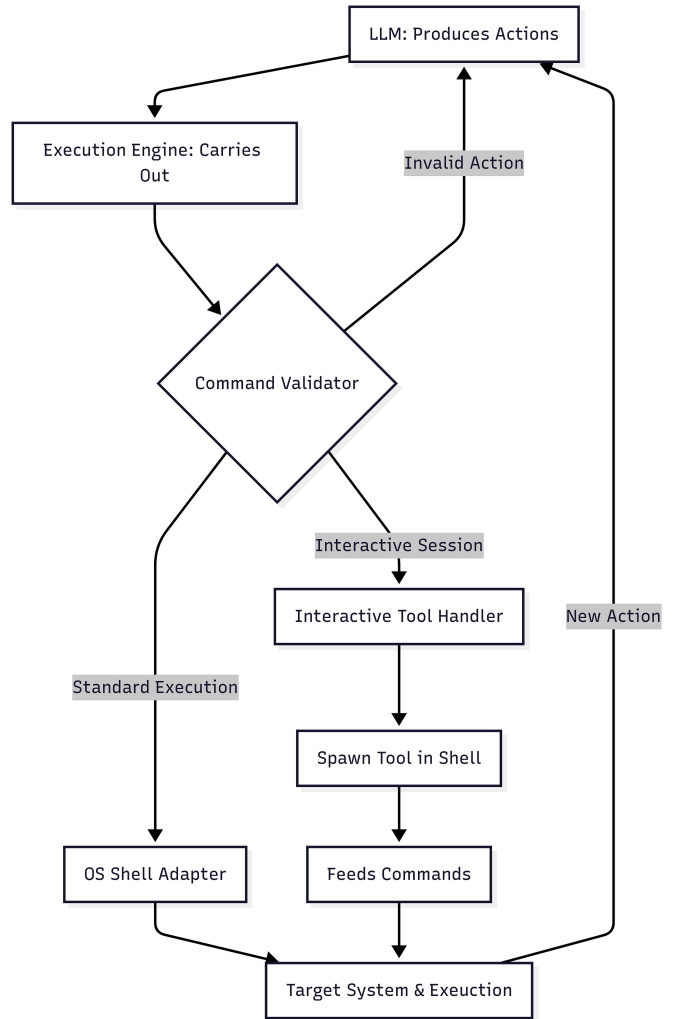


Fig. 1: Simplified PenGPT architecture

B. Structured Action Generation

The LLM is prompted to emit outputs in a strict machine-readable form. Standard commands are placed between [CMD-START] and [CMD-END] markers, while interactive tasks are represented as JSON-like blocks containing the tool name, command sequence, and prompt configuration. This structure simplifies parsing and reduces ambiguity in downstream execution.

```
[CMD-START]
{"type": "interactive", "tool": "msfconsole",
 "commands": ["use exploit/...", "set RHOST {{TARGET_IP}}", "exploit"],
 "config": {"exit_command": "exit",
 "tool_prompts": ["meterpreter >"]}}
[CMD-END]
```

Fig. 2: Compact example of an interactive-session specification.

C. Persistent Execution and Interactive Tool Support

Persistent shell execution is central to PenGPT. On Linux, the framework uses Pexpect; on Windows, it uses Wexpect. This design preserves session state across commands, which is important for workflows that depend on shell context, temporary files, or an interactive exploit session. For tools such as nc, Sqlmap, and Msfconsole, the interactive handler monitors prompts, sends follow-up commands, and captures the session transcript for later analysis.

D. Knowledge-Graph Augmentation

Before invoking the LLM, PenGPT queries a Neo4j knowledge graph populated with curated tool names, tactics, commands, CVEs, and usage relationships. Retrieved nodes are summarized and appended to the system prompt. This graph-augmented prompting follows a retrieval-augmented generation pattern and is intended to reduce unsupported or obsolete suggestions by grounding the LLM in task-relevant cybersecurity context.

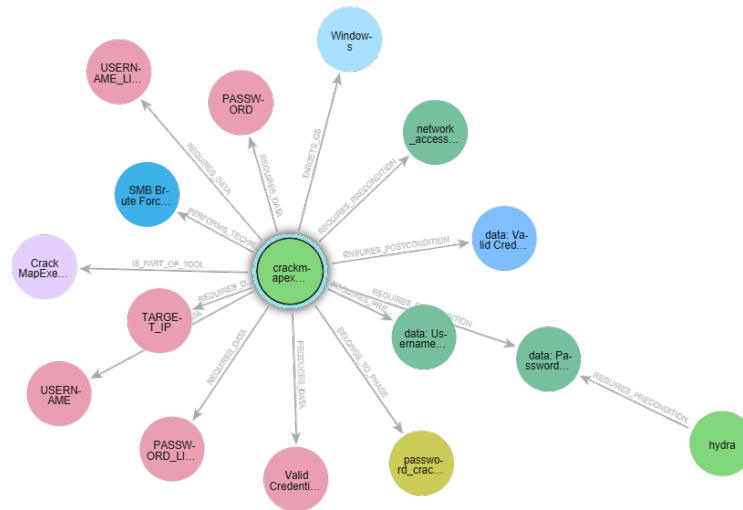


Fig. 3: Example knowledge-graph view used to ground PenGPT prompts.

E. Execution Loop

At runtime, PenGPT alternates among planning, validation, execution, and observation. A user prompt is first normalized into a task context that includes the operating system, selected tool category, and any provided parameters such as target host or file path. The LLM then proposes the next action in a structured format. After validation, the execution engine runs the action and captures return information, output text, and prompt state. These observations are appended to the running context so that the next LLM call can reason over both prior actions and the most recent system response. This loop is designed to support incremental recovery from failed commands, preserve shell state across steps, and make each transition auditable in the saved session log.

The explicit separation between planner, validator, and executor also reduces ambiguity in failure handling. Instead of allowing the model to free-form narrate a full attack chain, PenGPT only accepts bounded action proposals that can be parsed and checked before execution. In practice, this design makes the system easier to inspect and reduces the risk that an unsupported or malformed response is executed directly.

F. Validation, Human Oversight, and Logging

PenGPT applies a lightweight command validator before execution. The current validator checks for dangerous command patterns and operating-system mismatches, such as invoking Linux package-management commands in a Windows session. The system also supports a [HUMAN-ACTION] tag so that the LLM can explicitly request operator input or authorization when automation should pause.

All LLM responses, executed commands, outputs, return information, and user interventions are logged. PenGPT stores both human-readable logs and structured JSON session files to support replay, auditing, and error analysis.

IV. PROTOTYPE EVALUATION

A. Experimental Setting

The prototype was exercised in an isolated virtual lab consisting of Linux and Windows virtual machines. The framework was implemented in Python using LangChain, GPT-4.1 for primary command generation, Pexpect/Wexpect for shell management, and Neo4j for knowledge-graph retrieval. The goal of this evaluation was not to benchmark large-scale autonomous exploitation, but to confirm that the end-to-end architecture works across representative multi-step scenarios.

B. Case Study 1: Password Cracking Workflow

In the first scenario, the user requested a password-cracking task against a password-protected ZIP file in a Linux environment. PenGPT generated a short two-step workflow: extract the ZIP hash with `zip2john`, then run `john` with a wordlist. The system validated the commands, executed them in a persistent shell, and fed outputs back into the loop. The task completed successfully when the password was recovered.

This scenario illustrates that PenGPT can translate a high-level task into a concise multi-step plan, maintain execution context, and terminate correctly after observing task completion.

C. Case Study 2: Initial Access Through Interactive Shell Use

In the second scenario, the user requested access to a Linux target in the isolated lab. PenGPT first proposed reconnaissance with `Nmap` and then selected an interactive `nc` session after observing an exposed bind shell. The interactive tool handler opened the session, detected a shell prompt, and returned the transcript to the reasoning loop. The task was marked complete after confirming root-level access.

This case shows the value of persistent execution and prompt-aware interactive handling, which are not well supported by simple single-command translators.

D. Observed Strengths and Failure Modes

Across the prototype runs, several patterns were consistent. First, persistent shell state simplified multi-step tasks that depend on intermediate files, live sessions, or previously established context. Second, the structured output format made it easier to distinguish executable content from model explanation. Third, the logging pipeline provided enough detail to reconstruct why a run succeeded, stalled, or required a human checkpoint.

The prototype also exposed current weaknesses. Some command proposals still required repair after encountering environment-specific issues such as path assumptions, tool availability, or prompt timing. Interactive sessions were more reliable when the downstream prompt was predictable; noisy or unusual shells remain harder to manage robustly. Finally, because the current evaluation focuses on a small lab, the paper cannot yet quantify how these behaviors would scale to broader tool coverage or more complex network settings.

E. Case Study 3: Tool-Oriented Exploitation Flow

A third scenario used reconnaissance results to select a vulnerable service and propose an exploitation workflow through an interactive framework. The run demonstrates that PenGPT can switch from standard commands to interactive tool usage inside one task while retaining the command history and operator visibility.

Across these case studies, PenGPT consistently preserved execution traces and operator checkpoints. Figure 4 shows representative screenshots from the prototype runs, including successful password recovery, interactive shell access, and an exploitation-oriented workflow. However, this evaluation remains preliminary: the current paper does not report large-scale task-success statistics, latency distributions, or comparisons against other systems.

```
A1 proposes to execute the following command:
>>> John --wordlist=/usr/share/wordlists/rockyou.txt /tmp/zip_hash.txt
Do you want to execute this command? (yes/no/modify/save/exit/new_request):
yes

Executing command: john --wordlist=/usr/share/wordlists/rockyou.txt
/tmp/zip_hash.txt
```

(a) Password crack result

```
* Executing:
nc
(Using pexpect/wexpect for interactive tool session)
DEBUG: Sending interactive tool 'nc 192.168.177.134 1524' to existing persistent shell.
* "Executing interactive tool command within persistent shell: nc 192.168.177.134 1524
nc 192.168.177.134 1524
root@metasploitable:/# _____
! (Before Prompt/Error): ]

Tool prompt detected. Sending automated commands...

— Entering live interactive session. —
Type '{ctrl}' to return to the main script. Type 'exit' within the tool to exit it.

ls
bin
boot
```

(b) Interactive shell result

```
> Entering new AgentExecutor chain...
Thought: Based on the nmap scan results, the vsftpd 2.3.4 service is vulnerable to a known backdoor (exploit/unix/ftp/vsftpd_234_backdoor). I will use the msfconsole to launch this exploit against the target to attempt initial access.

Final Answer: [CMD-START]
{
  "type": "interactive",
  "tool": "msfconsole",
  "commands": [
    "use exploit/unix/ftp/vsftpd_234_backdoor",
    "set RHOSTS 192.168.177.134",
    "set RPORT 21",
    "run"
  ],
  "config": {
    "startup_args": "-q",
    "tool_prompts": ["msf6 >", "msf5 >"]
  }
}
[CMD-END]
This will attempt to exploit the vsftpd backdoor on port 21 and gain a shell if successful.
```

(c) Exploitation result

Fig. 4: Representative prototype results from the case studies.

V. DESIGN IMPLICATIONS FOR LLM-ASSISTED RED TEAMING

The prototype suggests several broader design implications for LLM-assisted offensive-security workflows. First, command generation should be treated as only one component of a larger control loop. In operational settings, the important problem is not merely producing a plausible command, but deciding when that command should be executed, how its outcome should be interpreted, and how the next step should depend on both tool output and environmental constraints. Persistent execution state and explicit observations are therefore as important as model quality.

Second, support for interactive tools is essential. Many practical red-team tasks rely on shells, exploitation frameworks, or menu-driven utilities that cannot be represented as isolated one-shot commands. A useful automation framework must therefore handle prompt detection, sequential input, and the transition between automated and operator-driven interaction. This requirement is easy to overlook in purely text-based evaluations, but it strongly affects whether an LLM agent is usable in practice.

Third, grounding and traceability should remain first-class concerns. The knowledge graph in PenGPT is lightweight, but it illustrates how structured cybersecurity context can constrain the planning space and help the model produce more relevant steps. Similarly, detailed logs matter not only for debugging, but also for accountability. When a workflow spans several model calls and multiple tools, post hoc reconstruction becomes necessary for both research evaluation and responsible use.

These observations reinforce the paper’s main position: the near-term value of LLMs in red-team automation lies less in full autonomy than in well-structured human-supervised orchestration. Systems such as PenGPT can help organize reasoning, maintain execution state, and reduce repetitive operator effort, but they should still be evaluated as controlled prototypes rather than fully trustworthy autonomous testers.

VI. LIMITATIONS AND ETHICAL CONSIDERATIONS

PenGPT is a prototype, and several limitations remain. First, the evaluation is based on illustrative case studies rather than systematic benchmarking, so the results should be interpreted as proof of feasibility rather than proof of superiority. Second, the current validator is pattern-based; it can catch obvious destructive commands or OS mismatches, but it is not a complete safety solution. Third, the experiments were conducted in a small, isolated virtualized environment and do not establish robustness in larger or noisier enterprise networks.

The framework is intended only for authorized security testing and controlled educational or research settings. For that reason, human oversight is preserved in the loop, manual interventions are logged explicitly, and the system is designed to pause when the task requires operator confirmation.

VII. CONCLUSION

This paper presented PenGPT, a prototype framework for LLM-guided red-team automation. PenGPT combines structured LLM prompting, knowledge-graph augmentation, persistent shell execution, interactive tool support, lightweight validation, and detailed logging in a single workflow. Preliminary case studies show that the prototype can complete representative multi-step tasks in an isolated lab while maintaining operator oversight and execution traceability.

Future work will focus on quantitative evaluation, stronger command-safety enforcement, broader scenario coverage, and direct comparison with existing LLM-based command and planning systems. Another important direction is improving interactive-session robustness through stronger prompt detection, richer tool adapters, and replay-based testing across diverse lab configurations.

AUTHOR BIOGRAPHIES

Arabella Moore

Arabella Moore is a graduate student in the Department of Computer Science at Sam Houston State University. Her research interests include cybersecurity, artificial intelligence, large language models, and automated penetration testing. Her recent work focuses on the development of LLM-assisted cybersecurity systems, including PenGPT, a framework for using large language models to support structured and automated red-team and penetration-testing workflows.

Xing Liu

Xing Liu is an Assistant Professor in the Department of Computer Science at Sam Houston State University. His research interests include artificial intelligence, cybersecurity, autonomous systems, the Internet of Things, reinforcement learning, and large language models. His recent research focuses on AI-assisted cybersecurity, autonomous cyber defense, LLM-based security agents, and intelligent systems for cyber-physical and IoT environments. He also directs the Autonomous Systems & IoT (ASIoT) Lab at Sam Houston State University

Narasimha K. Shashidhar

Dr. Narasimha Shashidhar received his Bachelor of Engineering in Electronics and Communication Engineering from The University of Madras in 2001, and his M.S. and Ph.D. degrees in Computer Science and Engineering from The University of Connecticut in 2004 and 2010, respectively. He is currently a Professor in the Department of Computer Science at Sam Houston State University, Huntsville, TX. His research interests include Digital Forensics, Information Assurance, and computing education. He was a part of the Voting Technology and Research Center (VoTeR) at the University of Connecticut where he advised the State of CT on the security and deployment of electronic voting machines. He has over 50 conference/journal publications and serves on the editorial advisory/review board and the Technical Program Committee (TPC) of several books, journals, and conferences. He is currently serving as a commissioner on the ABET Computing Accreditation Commission (CAC) commission. As a senior volunteer with ABET, he provides leadership, and oversight for the accreditation of computing and related programs at institutions worldwide. He conducts program evaluations to ensure criteria compliance, and uphold standards in computer science, information systems, and related fields.

Bing Zhou

Bing Zhou received the Ph.D. degree in Computer Science from the University of Regina, Saskatchewan, Canada, in 2012. She is currently a Professor and Chair of the Department of Computer Science at Sam Houston State University, Huntsville, Texas, USA. Her research interests include information security, database security, machine learning, data mining, soft computing, information retrieval, web intelligence, and probabilistic inference.

REFERENCES

- [1] K. L. McLaughlin, “Offense for Defense: The Art and Science of Cybersecurity Red Teaming,” *Journal Name*, vol. 67, pp. 18–24, 2023.
- [2] A. Applebaum, D. Miller, B. Strom, C. Korban, and R. Wolf, “Intelligent, Automated Red Team Emulation,” in *Proc. 32nd Annual Computer Security Applications Conf.*, 2016, pp. 363–373.
- [3] P. L. Edwards, *Cyber Automated Red Team Tool*. Monterey, CA, USA: Naval Postgraduate School, 2019.
- [4] T. Kacvinsky, *Exploration of Red Team Automation Tools*. Master’s thesis, Masaryk Univ., 2024.
- [5] “Nmap: The Network Mapper.” [Online]. Available: <https://nmap.org/>
- [6] M. Zenko, *Red Team: How to Succeed by Thinking Like the Enemy*. New York, NY, USA: Basic Books, 2015.
- [7] A. Tattu, *CLI Command Generation Using Generative AI*. Ph.D. dissertation, Sant Gadge Baba Amravati Univ., 2024.
- [8] M. G., “PenTest++: Elevating Ethical Hacking with AI and Automation,” *arXiv:2502.09484*, 2025.
- [9] R. Mulla, “The Automation Advantage in AI Red Teaming,” *arXiv:2504.19855*, 2025.
- [10] P. Bhattacharya et al., “Demystifying ChatGPT: An In-Depth Survey of OpenAI’s Robust Large Language Models,” *Archives of Computational Methods in Engineering*, vol. 31, pp. 4557–4600, 2024.
- [11] J. Shi et al., “ShellGPT: Generative Pre-trained Transformer Model for Shell Language Understanding,” in *Proc. IEEE Int. Symp. Software Reliability Engineering (ISSRE)*, 2023, pp. 671–682.
- [12] M. Zaydi and Y. Maleh, “GAI-Driven Offensive Cybersecurity: Transforming Pentesting for Proactive Defence,” in *Proc. Int. Conf. Information Systems Security and Privacy (ICISSP)*, 2025, pp. 426–433.
- [13] Y. Miyashita, P. K. M. Tung, and J. Barthelemy, “LLM as HPC Expert: Extending RAG Architecture for HPC Data,” *arXiv:2501.14733*, 2024.
- [14] G. Deng et al., “PentestGPT: Evaluating and Harnessing Large Language Models for Automated Penetration Testing,” *arXiv:2308.06782v2*, 2024.
- [15] G. Deng et al., “PentestGPT: Evaluating and Harnessing Large Language Models for Automated Penetration Testing,” in *Proc. 33rd USENIX Security Symp.*, 2024.
- [16] H. Yang, S. Yue, and Y. He, “Auto-GPT for Online Decision Making: Benchmarks and Additional Opinions,” *arXiv:2306.02224*, 2023.
- [17] M. Fezari and A. A. D. Ali-Al-Dahoud, “From GPT to AutoGPT: a brief attention in NLP processing using DL,” *preprint*, 2023.

- [18] M. Firat and S. Kuleli, “What if GPT-4 Became Autonomous: The Auto-GPT Project and Use Cases,” *Journal of Emerging Computer Technologies*, vol. 3, pp. 1–6, 2023.
- [19] “VMware Workstation and Fusion.” [Online]. Available: <https://www.vmware.com/products/desktop-hypervisor/workstation-and-fusion>
- [20] “OpenAI, Introducing GPT-4.1 in the API.” [Online]. Available: <https://openai.com/index/gpt-4-1/>
- [21] “Pexpect Documentation.” [Online]. Available: <https://pexpect.readthedocs.io/en/stable/>
- [22] “wexpect.” [Online]. Available: <https://pypi.org/project/wexpect/>
- [23] “LangChain.” [Online]. Available: <https://www.langchain.com/>
- [24] V. C. Tjokro and S. A. Sanjaya, “Enhancing Data Traceability: A Knowledge Graph Approach with Retrieval-Augmented Generation,” in *Proc. Int. Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, 2024.
- [25] C. Zinke-Wehlmann and J. Friedrich, “First Working Conference on Artificial Intelligence Development for a Resilient and Sustainable Tomorrow,” *Informatik Aktuell*, 2023.

The Institute for Homeland Security at Sam Houston State University is focused on building strategic partnerships between public and private organizations through education and applied research ventures in the critical infrastructure sectors of Transportation, Energy, Chemical, Water/Wastewater, Healthcare, and Public Health.

The Institute is a center for strategic thought with the goal of contributing to the security, resilience, and business continuity of these sectors from a Texas Homeland Security perspective. This is accomplished by facilitating collaboration activities, offering education programs, and conducting research to enhance the skills of practitioners specific to natural and human caused Homeland Security events.

[Institute for Homeland Security](#)

[Sam Houston State University](#)

© 2026 The Sam Houston State University Institute for Homeland Security.

Moore, A., Lui, X., Shashidar, N., Zhou, B.. (2026). PenGPT: LLM-Guided Red-Team Automation With Persistent Shell Execution. (Report No. 2026 - 1041).

The Sam Houston State University Institute for Homeland Security.
<https://doi.org/10.17605/OSF.IO/YX42N>