Construction and Evaluation of LLM-based agents for Semi-Autonomous penetration testing

Masaya Kobayashi [1] **Masane Fuchi** [1] **Amar Zanashir** [2] **Tomonori Yoneda** [2] **Tomohiro Takagi** [1]

Abstract

With the emergence of high-performance large language models (LLMs) such as GPT, Claude, and Gemini, the autonomous and semi-autonomous execution of tasks has significantly advanced across various domains. However, in highly specialized fields such as cybersecurity, full autonomy remains a challenge. This difficulty primarily stems from the limitations of LLMs in reasoning capabilities and domain-specific knowledge. We propose a system that semi-autonomously executes complex cybersecurity workflows by employing multiple LLMs modules to formulate attack strategies, generate commands, and analyze results, thereby addressing the aforementioned challenges. In our experiments using Hack The Box virtual machines, we confirmed that our system can autonomously construct attack strategies, issue appropriate commands, and automate certain processes, thereby reducing the need for manual intervention.

1. Introduction

The real-world applications of large language models (LLMs) are increasing, with coding assistance and business process autonomy being representative examples. LLMs possess the ability to execute complex tasks on the basis of text-based instructions, and their potential has been demonstrated in various application scenarios (Wu et al., 2023; Wang et al., 2023; Chan et al., 2023; Wang et al., 2024).

In the field of cybersecurity, LLMs have been applied to tasks such as vulnerability and phishing detection, autonomy and diversification of attacks, and information manipulation (Yao et al., 2024). Notably, dynamic and flexible approaches using LLMs have demonstrated performance comparable with or exceeding traditional static methods in areas such as vulnerability discovery through fuzzing, code remediation, and the creation of targeted phishing emails. (Noever, 2023; Koide et al., 2025; Xia et al., 2024) However, in attack simulations and vulnerability assessments of operating systems, LLMs face challenges in adapting to environments and acquiring and utilizing specialized knowledge, limiting their effectiveness.

We focus on penetration testing and explore methods for leveraging LLMs to achieve semi-autonomy. Research on LLM-assisted penetration testing includes expert support systems for strategy formulation and execution planning (Deng et al., 2024), autonomy focused on post-exploitation (Xu et al., 2024), privilege escalation (Happe & Cito, 2023; Happe et al., 2024), web-based attacks (Fang et al., 2024b), zero-day attack autonomy (Fang et al., 2024a), and one-day attack autonomy (Fang et al., 2024c). However, these studies are limited in terms of the attack phases they automate. Research that attempts to automate the entire penetration testing process (Huang & Zhu, 2023) relies on specific environments and tools, which impose numerous operational constraints, making general application in real-world scenarios challenging.

Considering real-world applications, a less restrictive system is required. Therefore, we propose a novel system for autonomous penetration testing. Inspired by recent advancements in prompt engineering, such as Self-Refine (Madaan et al., 2023) and ReAct (Yao et al., 2023), we design an LLM-based agent architecture and integrate information retrieval techniques, such as retrieval-augmented generation (RAG) (Lewis et al., 2020), to efficiently utilize specialized knowledge for strategy formulation and execution planning.

In our experiments using Hack The Box (noa, c) virtual machines, the results demonstrate that our system can autonomously construct attack strategies, generate appropriate commands, and automate certain processes, significantly reducing the need for manual intervention. However, since most existing approaches are not publicly available, direct comparisons are not possible, and quantitative evaluation using benchmarks has yet to be conducted. Thus, the evaluation of our system remains qualitative. Nevertheless, experimental results suggest that our system overcomes the

[1]Meiji University, Kanagawa, Japan [2]LAC Co. Ltd., Tokyo, Japan. Correspondence to: Masaya Kobayashi <msy.kbysh02@gmail.com>.

limitations of conventional methods and offers a flexible, autonomous penetration testing framework capable of adapting to diverse attack scenarios.

2. Backgrounds

2.1. Penetration Testing

Penetration testing is a method for assessing system vulnerabilities from the perspective of an actual attacker, traditionally conducted manually by security experts. In recent years, various autonomous approaches utilizing LLMs have been proposed, including support systems (Deng et al., 2024), autonomous systems focusing on specific attack stages (Xu et al., 2024; Happe & Cito, 2023), systems for automating vulnerability exploitation (Fang et al., 2024a;c), and frameworks that attempt to autmate the entire penetration testing process (Huang & Zhu, 2023). These approaches aim to autonomously generate and execute attack scenarios, which is difficult with traditional methods. However, a common challenge among them is that they have not yet achieved full autonomy.

2.2. Retrieval-augmented Generation

Retrieval-augmented generation (Lewis et al., 2020; Gao et al., 2024) is a technique that dynamically incorporates external knowledge during text generation, overcoming the limitation of traditional LLMs relying solely on knowledge acquired during training. A representative method, Hybrid Search (Łukawski), integrates keyword-based search with semantic vector search to enhance information retrieval accuracy. Additionally, Reranking (noa, h) is used to select the most relevant information from retrieved data. These methods enable the generation of responses that reflect the latest information and specialized domain knowledge.

2.3. Prompt Engineering

Prompt engineering is a technique for designing input prompts to optimize LLMs' output. Chain-of-Thought prompting (Wei et al., 2023) facilitates step-by-step reasoning for complex problems by leveraging internal knowledge. On the other hand, Self-Refine (Madaan et al., 2023) enables an LLM to generate feedback on its initial output and iteratively refine it on the basis of that feedback. Additionally, ReAct (Yao et al., 2023) is a prompting method that alternates between reasoning and action generation, enabling the model to dynamically incorporate external information while planning and executing high-level tasks. These techniques enhance the reasoning capabilities of LLMs in complex security tasks without requiring additional training.

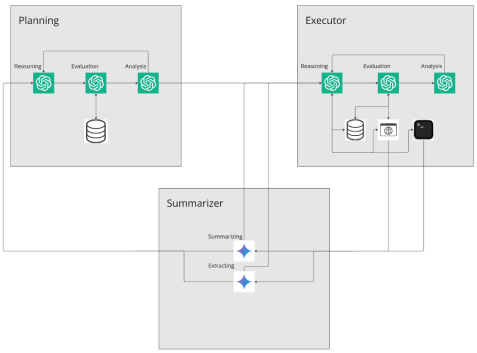

Figure 1. Overview of our proposed method.

2.4. LLM-based Agents

LLM-based agents leverage the powerful generative capabilities of LLMs to autonomously perform decision-making, planning, interaction, and integration with external environments beyond simple text generation. Traditional LLMs have been widely used as static tools that generate responses on the basis of given inputs (Kojima et al., 2023; Brown et al., 2020). However, LLM-based agents extend this capability to handle complex problem-solving and interactive tasks (Wu et al., 2023; Wang et al., 2023; Chan et al., 2023). By integrating internal reasoning with external actions, LLM-based agents enable more transparent and sophisticated decision-making processes.

3. Proposed Method

3.1. Overview

Our semi-autonomous penetration testing system consists of multiple LLMs, each assigned distinct roles in reasoning, evaluation, and analysis. These LLMs are organized into three modules: planning, execution, and summarization. Additionally, the system integrates RAG, search engines, and an execution environment as tools. Figure 1 provides an overview. In this system, human intervention is minimized to decision-making and monitoring of command execution. The system is designed to be independent of specific environments, enabling penetration testing using tools executable on the Kali Linux command line.

3.2. Planning Module

The planning module formulates attack strategies for penetration testing and manages the entire intrusion process. It employs the Pentesting Task Tree (PTT), a data structure used in related research, to track attack phases. On the basis of the summarization module's results, the system updates the state and information of tasks and subtasks for each attack phase. The PTT structure in this study is built upon the

MITRE ATT&CK framework (noa, d) and ensures that all recorded execution results are accessible to every module.

The module's architecture draws inspiration from Self-Refine and ReAct. The roles of PTT generation and updating (reasoning), feedback provision on inference results (evaluation), and information retrieval and improvement proposal generation (analysis) are distributed among multiple LLMs. This iterative process refines the PTT. Furthermore, by incorporating execution results, the system ensures the generation of accurate execution procedures.

3.3. Execution Module

The execution module translates attack strategies into concrete execution steps and generates terminal-executable commands. Similar to the planning module, commands undergo refinement through the reasoning, evaluation, and analysis processes of multiple LLMs. Human operators review the generated commands and explanations to determine whether to execute them in the terminal while monitoring the process. This intervention is necessary because LLMs remain susceptible to hallucinations, and some operations—such as those requiring root privileges or actions that may compromise system integrity—demand human oversight.

3.4. Summarization Module

The summarization module consists of LLMs dedicated to summarization and extraction. The summaries of execution and search results are strictly on the basis of retrieved information, excluding any speculative reasoning or predictions from the LLMs. The extraction process identifies and isolates critical data, such as passwords and authentication credentials. This ensures that even if the execution results contain redundant information, only essential details are passed on to the planning and execution modules, improving efficiency and clarity in decision-making.

3.5. Tools

3.5.1. Information Gathering

For LLMs to effectively leverage specialized knowledge, efficient information gathering is essential. We employ a combination of RAG and web searches to collect relevant data. When the agent determines that additional information is necessary, it selects an appropriate tool and generates a search query accordingly.

For RAG, useful penetration testing resources such as MITRE (noa, d), OWASP (noa, f), and references for Kali Linux Tools (noa, b) are stored in a Qdrant (noa, g) vector database. Relevant documents are retrieved on demand.

For web searches, the system can extract information from sources such as the National Vulnerability Database (NVD) (nvd) and publicly available repositories on GitHub (git). This approach provides access to the latest security knowledge.

3.5.2. Execution Environment

The execution environment enables multiple terminal operations using subprocess (Python) and tmux (Linux). On the basis of execution procedures generated by the execution module, specified commands are executed in designated terminal sessions.

4. Experiment

4.1. Experimental Environment

We conducted experiments on virtual machines provided by Hack The Box. The target machine was selected on the basis of the criteria that the official Hack The Box difficulty rating was "Easy" and the user rating was more than 4.5, with a preference for machines incorporating a relatively large number of technical elements. On the basis of these criteria, we selected Board Light[1] as the test target. Table 1 presents the testing methods used in our experiment.

For our proposed method, we used GPT-4o (OpenAI et al., 2024) for the planning and execution modules, while Gemini-1.5-flash (Team et al., 2024) was utilized for the summarization module. Additionally, for the vector database, text-embedding-3-large (ope, 2024) was used as the embedding model, and Rerank 3 (coh) was employed as the reranking model.

4.2. Results and Discussion

Table 2 presents a subset of the tasks and commands generated during the test (only relevant sections are extracted). Our system successfully performed an autonomous penetration testing, except for the part involving directory and file exploration within the server during a password attack.

We found that in the planning module, initial inference results sometimes failed to generate the necessary strategies. However, through evaluation and analysis, the system was able to refine and correct these strategies. This demonstrates that LLMs that engage in iterative reasoning cycles are effective in deepening their insights. Table 3 compares the initial and refined generations of the planning module. In this table, the initial plan only included an attack strategy targeting SSH (port 22), whereas after evaluation, an additional strategy for HTTP (port 80) was also generated.

In the execution module, tasks that could be executed with

[1] https://www.hackthebox.com/machines/boardlight

Table 1. Overview of Board Light

Element	Description
Port Scanning	Port scanning using Nmap
Web Application Analysis	Information gathering using WhatWeb and Curl Subdomain and directory enumeration using Ffuf and Gobuster
Initial Exploitation	Exploit utilizing CVE-2023-30253
Authentication	Privilege escalation to another user on the basis of authentication credentials
Privilege Escalation	Information gathering using Linpeas Exploit utilizing CVE-2022-37706

a single command were mostly generated correctly on the first attempt through reasoning and analysis. Table 4 compares the initial and refined generations of the execution module. The initial command generation did not correctly utilize user credentials or passwords, but after refinement, the execution steps were corrected on the basis of errors and retrieved information. However, when multiple commands were required such as retrieving and executing an exploit script from GitHub,— errors occurred several times. To address this, the system dynamically executed web searches on the basis of the generated queries, retrieved references such as README files, and refined the commands accordingly. This highlights the effectiveness of tools that supplement specialized knowledge, such as Kali Linux references. This ability to utilize tools and process large amounts of information is a key strength of LLM-based agents, enabling autonomous penetration testing for known vulnerabilities.

However, when conducting post-exploitation tasks such as exploring directories and files within the compromised server, although the system selected appropriate tools, it demonstrated poor sensitivity to environment-specific details such as directory names and file names. The extensive trial-and-error process required for this exploration made task completion more challenging. Furthermore, unlike other tools, this process relies on internal system information rather than external sources, highlighting a limitation of information retrieval tools that do not guarantee accurate data. This also suggests the limitations of applying a generalized LLMs to every stage of a penetration testing.

To address these challenges in the future, we believe that adopting a knowledge graph instead of a vector database for information retention and creating specialized agents tailored to each attack phase would be effective. Assigning dedicated agents to specific tasks would facilitate the development of a more advanced LLM-based agent system.

5. Conclusion and Future Work

We proposed an LLM-based agent system utilizing multiple LLM modules to address the challenges of applying LLMs to autonomous penetration testing, specifically the lack of specialized knowledge and the complexity of reasoning. Through experiments, we confirmed that the system can autonomously conduct penetration testing by leveraging complex reasoning across multiple LLMs and flexible information retrieval. However, we also identified limitations, particularly in exploratory approaches to environment-specific configurations.

Additionally, due to the difficulty of enabling LLMs to operate the GUI of penetration testing tools, we constrained our implementation to command-line executable tools. However, by integrating systems such as Anthropic's Computer Use (noa, a) and OpenAI's Operator (noa, e), it may be possible to develop a more advanced LLM-based agent system capable of executing even more sophisticated penetration testing tasks.

We plan to continue refining our approach by addressing the limitations identified in this study and improving the overall system architecture.

References

Introducing Rerank 3: A New Foundation Model for Efficient Enterprise Search & Retrieval. URL https://cohere.com/blog/rerank-3.

GitHub Japan. URL https://github.co.jp/.

Computer use (beta), a. URL https://docs.anthropic.com/en/docs/build-with-claude/computer-use.

Get Kali, b. URL https://www.kali.org/get-kali/.

Hack The Box: The #1 Cybersecurity Performance Center, c. URL https://www.hackthebox.com.

MITRE ATT&CK®, d. URL https://attack.mitre.org/.

Introducing operator, e. URL https://openai.com/index/introducing-operator/.

Table 2. Planning and executing module results

Generated task	Generated command
Perform a port scan	nmap -sS -sV 10.10.11.11
Web Information Gathering	thttp://10.10.11whatweb –a 3 –v thttp://10.10.11.11 curl –I
Subdomain enumeration for "board.htb"	gobuster dns –d board.htb –w bitquark–subdomains–top100000.txt thttp://10.10.11.11 ffuf –w subdomains–top1million–5000.txt –u –H "Host:FUZZ.crm.board.htb"
Web technology stack identification for subdomains	whatweb -a 3 http://crm.board.htb
Search for CVEs and exploit codes on NVD and GitHub	search: Dolibarr version17.0.0 CVE search: site:github.com Dolibarr 17.0.0 exploit search: site:nvd.nist.gov Dolibarr 17.0.0 CVE
Exploit remote command execution and file upload vulnerabilities in Dolibarr	git clone https://github.com/nikn0laty/Exploit–for–Dolibarr–17.0.0–CVE–2023–30253.git cd Exploit–for–Dolibarr–17.0.0–CVE–2023–30253 nc –lnvp 4444 python3 exploit.py crm.board.htb admin admin 10.10.14.2 4444
Search for configuration files	find / –type f (–name "∗.conf" –o –name "∗.ini" –o –name "∗.env" –o –name "∗.yaml" –o –name "∗.php
Identify password files and sensitive information	cat /etc/password find / –perm –u=s –type f 2>/dev/null
Attempt SSH connections with discovered credentials	sshpass –p 'serverfun2$2023!!' ssh larissa@10.10.11.11 –p 22
Search for additional vulnerabilities	enlightment –version
Exploit system services for privilege escalation	search: enlightment 0.23.1 CVE git clone https://github.com/MaherAzzouzi/CVE–2022–37706–LPE–exploit cd CVE–2022–37706–LPE–exploit scp exploit.sh larissa@10.10.11.11:exploit.sh ./exploit.sh

Table 3. Capabilities of the planning module (partial list)

Initial Generation	Modification
1 Reconnaissance – [completed] ... 1.2 Identify Open Ports and Services – (completed) 1.2.1 Perform a port scan – (completed) – result: – ∗∗Open Ports∗∗: 22/tcp (ssh), 80/tcp (http) – ∗∗Services∗∗: – 22/tcp: ssh OpenSSH 8.2p1 Ubuntu 4ubuntu0.11 (Ubuntu Linux; protocol 2.0) – 80/tcp: http Apache httpd 2.4.41 ((Ubuntu)) – ∗∗OS∗∗: Linux (detected by nmap) – ∗∗Network Distance∗∗: 2 hops – ∗∗SSH Host Keys∗∗: – RSA: 3072 ... – ECDSA: ... – ED25519: 256 ... 1.2.2 Detailed scan on specific ports – (completed) – result: – ∗∗Service Info∗∗: OS: Linux; CPE: cpe:/o:linux:linux_kernel – ∗∗HTTP Server Header∗∗: Apache/2.4.41 (Ubuntu) – ∗∗HTTP Title∗∗: Site doesn't have a title (text/html; charset=UTF–8). 2 Initial Access – [todo] 2.1 Exploit Discovered Services – (todo) 2.1.1 Attempt SSH Access – (todo) 2.1.2 Investigate HTTP Service for Vulnerabilities – (todo)	1 Reconnaissance – [completed] ... 1.3 Web Information Gathering – [todo] 1.3.1 Identify web technologies and frameworks – (todo) 1.3.2 Enumerate web directories and files – (todo) 1.3.3 Analyze web content for sensitive information or misconfigurations – (todo) 2 Initial Access – [todo] ... 2.2 Vulnerability Assessment – [todo] 2.2.1 Check for known CVEs related to Apache and OpenSSH – (todo) 2.2.2 Investigate potential misconfigurations or outdated software versions – (todo)

Table 4. Capabilities of the execution module (selected excerpts)

Initial Generation	Modification
git clone https :// github .com /... cd ... thttp://crm.board.htbpython3 exploit .py < USERNAME> <PASSWORD> 10.10.14.2 4444	git clone https :// github .com /... cd ... thttp://crm.board.htbpython3 exploit .py admin admin 10.10.14.2 4444

OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation, f. URL https://owasp.org/.

Qdrant - Vector Database, g. URL https://qdrant.tech/.

Reranking in Semantic Search - Qdrant, h. URL https://qdrant.tech/documentation/search-precision/reranking-semantic-search/.

NVD - Home. URL https://nvd.nist.gov/.

Vector embeddings, March 2024. URL https://platform.openai.com/docs/guides/embeddings.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Łukawski, K. Hybrid Search Revamped - Building with Qdrant's Query API - Qdrant. URL https://qdrant.tech/articles/hybrid-search/.

Chan, C.-M., Chen, W., Su, Y., Yu, J., Xue, W., Zhang, S., Fu, J., and Liu, Z. Chateval: Towards better llm-based evaluators through multi-agent debate, 2023. URL https://arxiv.org/abs/2308.07201.

Deng, G., Liu, Y., Vilches, V. M., Liu, P., Li, Y., Xu, Y., Pinzger, M., Rass, S., Zhang, T., and Liu, Y. Pentestgpt: Evaluating and harnessing large language models for automated penetration testing. In *USENIX Security Symposium*, 2024. URL https://www.usenix.org/conference/usenixsecurity24/presentation/deng.

Fang, R., Bindu, R., Gupta, A., and Kang, D. Llm agents can autonomously exploit one-day vulnerabilities, 2024a. URL https://arxiv.org/abs/2404.08144.

Fang, R., Bindu, R., Gupta, A., Zhan, Q., and Kang, D. Llm agents can autonomously hack websites, 2024b. URL https://arxiv.org/abs/2402.06664.

Fang, R., Bindu, R., Gupta, A., Zhan, Q., and Kang, D. Teams of llm agents can exploit zero-day vulnerabilities, 2024c. URL https://arxiv.org/abs/2406.01637.

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., and Wang, H. Retrieval-augmented generation for large language models: A survey, 2024. URL https://arxiv.org/abs/2312.10997.

Happe, A. and 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. ACM, November 2023. doi: 10.1145/3611643.3613083. URL http://dx.doi.org/10.1145/3611643.3613083.

Happe, A., Kaplan, A., and Cito, J. Llms as hackers: Autonomous linux privilege escalation attacks, 2024. URL https://arxiv.org/abs/2310.11409.

Huang, J. and 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, 2023.

Koide, T., Fukushi, N., Nakano, H., and Chiba, D. Detecting phishing sites using chatgpt, 2025. URL https://arxiv.org/abs/2306.05816.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners, 2023. URL https://arxiv.org/abs/2205.11916.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. Retrieval-augmented generation for knowledge-intensive nlp tasks. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B. P., Hermann, K., Welleck,

S., Yazdanbakhsh, A., and Clark, P. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=S37hOerQLB.

Noever, D. Can large language models find and fix vulnerable software?, 2023. URL https://arxiv.org/abs/2308.10345.

OpenAI, :, Hurst, A., Lerer, A., Goucher, A. P., Perelman, A., et al. Gpt-4o system card, 2024. URL https://arxiv.org/abs/2410.21276.

Team, G., Georgiev, P., Lei, V. I., Burnell, R., Bai, L., et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context, 2024. URL https://arxiv.org/abs/2403.05530.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. Voyager: An open-ended embodied agent with large language models, 2023. URL https://arxiv.org/abs/2305.16291.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., and Wen, J. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6), March 2024. ISSN 2095-2236. doi: 10.1007/s11704-024-40231-1. URL http://dx.doi.org/10.1007/s11704-024-40231-1.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., and Wang, C. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023. URL https://arxiv.org/abs/2308.08155.

Xia, C. S., Paltenghi, M., Le Tian, J., Pradel, M., and Zhang, L. Fuzz4all: Universal fuzzing with large language models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, pp. 1–13. ACM, April 2024. doi: 10.1145/3597503.3639121. URL http://dx.doi.org/10.1145/3597503.3639121.

Xu, J., Stokes, J. W., McDonald, G., Bai, X., Marshall, D., Wang, S., Swaminathan, A., and Li, Z. Autoattacker: A large language model guided system to implement automatic cyber-attacks. *arXiv preprint arXiv:2403.01038*, 2024.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

Yao, Y., Duan, J., Xu, K., Cai, Y., Sun, Z., and 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, June 2024. ISSN 2667-2952. doi: 10.1016/j.hcc.2024.100211. URL http://dx.doi.org/10.1016/j.hcc.2024.100211.