(Special Session)

LLMs for Hardware Security: Boon or Bane?

Rahul Kande, Vasudev Gohil, Matthew DeLorenzo, Chen Chen, Jeyavijayan Rajendran
Texas A&M University, USA
{rahulkande, gohil.vasudev, matthewdelorenzo, chenc, jv.rajendran}@tamu.edu

2024 IEEE 42nd VLSI Test Symposium (VTS) | 979-8-3503-6378-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/VTS60656.2024.10538871

***Abstract*—Large language models (LLMs) have emerged as transformative tools within the hardware design and verification lifecycle, offering numerous capabilities in accelerating design processes. Recent research has showcased the efficacy of LLMs in translating design specifications into source code through hardware description languages. Researchers are also using LLMs to generate test cases and write assertion rules to bolster the detection of hardware vulnerabilities. Thus, the semiconductor industry is swiftly integrating LLMs into its design workflows. However, this adoption is not without its challenges.**

While LLMs offer remarkable benefits, they concurrently introduce security concerns that demand a thorough examination. These concerns manifest as potential vulnerabilities indirectly introduced into the designs while generating the design code, or by directly equipping the attackers with novel avenues for exploitation. In this paper, we discuss the emerging security implications due to the capabilities introduced by LLMs in the context of hardware design verification, evaluate the capabilities of existing security detection and mitigation techniques, and highlight the possible future security attacks that use LLMs.

***Index Terms*—LLM, hardware, verification, security**

I. Introduction

A. Hardware Design Verification

Hardware verification techniques can be primarily classified into regression-based testing and formal verification. Hardware regression-based testing involves repeatedly generating and simulating input tests with the target design (e.g., random regression [1]). Most regression-based techniques use a golden reference model or assertions inserted into the hardware to detect vulnerabilities [2]. Formal verification techniques are widely applied in the industry [3], [4], which explore design spaces exhaustively and prove if a design-under-test (DUT) satisfies specified properties [5]. Based on security specifications, designers write assertions and use commercial formal tools to prove if the design violates any requirements. While regression-based techniques scale to large designs, they suffer from their slow design coverage in detecting vulnerabilities. On the other hand, formal techniques can fully verify the given functional and security properties, but do not scale well to large and complex modern hardware. Also, writing properties requires expert knowledge of the DUT, which is error-prone and time-consuming [6], [7].

Recently, hardware fuzzing has shown its effectiveness in detecting vulnerabilities in large-scale designs such as modern processors [8]–[11]. Hardware fuzzing techniques generate test cases dynamically and analyze the execution logs of a DUT to see if the design violates any assertions [12] or triggers vulnerabilities [8]–[10], [13]–[15]. However, fuzzers require time and compute resources to execute the hardware, which is precious given the market demand to shrink verification time.

B. Supply-Chain Security

The globalization of the integrated circuit (IC) supply chain has brought significant benefits to the production of electronic products, such as cost reductions, access to specialized expertise, and quicker market entry. However, it has also introduced complex hardware security issues that undermine trust in the IC ecosystem, with the threat of hardware Trojans (HTs) during IC fabrication being a particularly alarming concern. These malicious modifications, covertly inserted during the manufacturing process, can severely compromise the integrity and functionality of ICs upon activation, facilitating a variety of cyberattacks. Such attacks can alter an IC's operations, degrade performance, expose sensitive data, or even disable entire systems, necessitating sophisticated security strategies for their detection and neutralization [16].

IC security issues persist even after a chip has been manufactured, exposing it to multiple forms of attack. Among these, fault injection methods, including voltage and clock glitching, have become a significant threat [17]. These techniques can severely impact the operational reliability and security of ICs, leading to dire outcomes. In recent years, reinforcement learning (RL) has emerged as an effective strategy to address these hardware security concerns. Researchers have devised various RL formulations to detect hardware Trojans [18]–[20] and evaluate defenses [21], assess vulnerability of ciphers to fault injection attacks [22], and even to evaluate the robustness of machine learning based HT detection techniques against attacks [23]. However, these RL-based approaches require (i) extensive training processes, which are time- and compute-intensive, and (ii) a deep understanding of the target problems with steep learning curves.

C. Large Language Models

Large Language Models (LLMs) have revolutionized various fields of natural language processing and code-related tasks. In natural language processing, LLMs excel in areas including language translation [24], sentiment analysis [25], summarization, question answering, and more, owing to their ability to understand and generate human-like text. Moreover, LLMs can generate code snippets or even entire programs based on natural language descriptions, facilitating rapid prototyping and code design. In code completion tasks, they provide intelligent suggestions for completing code segments, improving developer productivity, and reducing errors [26].

Overall, LLMs represent a powerful tool for enhancing natural language processing and code-related tasks, offering versatility and efficiency across various applications.

D. Hardware Security in the Context of LLMs

In the realm of hardware security, LLMs are emerging as pivotal assets, shaping academic research and industry practices in hardware design and verification workflow [27]–[29]. LLM applications in hardware design span from code completion [28] to code correction [30]. Furthermore, LLMs are undergoing continuous fine-tuning to enhance their performance in comprehending and generating hardware description language (HDL) code, amplifying their role in aiding hardware design [28], [31], [32]. These capabilities of LLMs are evident through the shifting trends in semiconductor industries that are adopting LLMs in their electronic design automation (EDA) flow [33]. Beyond hardware design, LLMs are also starting to play a major role in hardware verification to detect functional and security vulnerabilities, including detecting hardware trojans and faults (see Sec. II).

However, as LLMs augment design and verification tasks, they are also a cause for concern, as they can potentially introduce new security vulnerabilities and attack surfaces into hardware. Existing hardware verification techniques can address some concerns, like vulnerabilities inserted by LLMs during code completion. However, the capabilities of LLMs can result in new attacks, necessitating novel security defenses to safeguard against emerging threats. Therefore, while leveraging LLMs' potential for hardware design and verification, it is imperative to concurrently address the evolving security landscape and potential attack vectors introduced by these powerful models (see Sec. III).

II. LLMs for Hardware Verification

Other than generating hardware code, LLMs are also aiding with the verification of hardware as we discuss below.

A. Analyzing Code to Aid Debugging

Given the ability of LLMs to analyze and generate text (Sec. I), LLMs can be utilized as a method for detecting code vulnerabilities. One strategy includes utilizing in-context learning, in which an LLM is provided a few ideal examples (input-output pairs) before being prompted with a test input (i.e. code to analyze). This strategy enabled ChatGPT-3.5 to perform competitively with the fine-tuned CodeBERT in determining if tested codes contain a vulnerability, with an accuracy of up to 62.7% [34]. LLMs have also demonstrated the ability to debug hardware (HDL). Through fine-tuning an LLM with a dataset of defective hardware designs, the ability of LLMs to effectively identify and correct defects within Verilog programs increased [35]. However, even advanced language models (GPT-4, PaLM2), are often unreliable when prompted to identify specific vulnerabilities within source codes, as the LLM may provide responses that are based upon incorrect reasoning, non-deterministic, or not robust [36].

LLMs can also aid the development process at the IC level, such as in writing architecture specifications. To minimize the human error and time associated with this task, specific LLM prompting strategies can result in the successful generation of effective specifications, ranging from high- to low-level architectural descriptions [37]. Furthermore, RTL code can be directly utilized in this prompting framework, providing a confined scope for the specification. Additionally, the text analysis capabilities of LLMs enable the ability to generate an effective overview of a provided specification, even containing additional unique insights [37].

B. Generating Test Cases Using LLMs

Detecting vulnerabilities entails generating test inputs that trigger activity in various hardware components, thereby revealing the vulnerabilities present. While techniques like fuzzing use mutation strategies to explore the design space, they still use randomly generated inputs as the starting points [8]. LLMs, however, can understand a given hardware design and generate test inputs targeting various regions of hardware [38]. These LLM-generated test cases, coupled with mutations created by the fuzzer, can increase the overall coverage and accelerate vulnerability detection.

C. Generating Assertions Using LLMs

Both regression-based and formal verification techniques use assertions, i.e., functional and security properties, to detect vulnerabilities [39]. These properties define the expected behavior of the hardware designs, from which any deviation is flagged as a vulnerability. Assertions are effective in verifying vulnerabilities at the early stages of design, unlike reference model-based detection techniques, since assertions can be verified on individual design modules and do not require the output of the entire design. These properties are developed either manually by referring to specifications or mined from the existing hardware designs. However, manual creation of the assertions consumes time and requires design expertise, while mining the properties requires existing hardware designs [6], [7]. This limitation in the generation of assertions limits the adaption of assertions into hardware verification workflows.

Using LLMs to generate hardware assertions resolves this limitation, as LLMs enable faster and more automated generation of assertions to detect various hardware vulnerabilities [40]–[44]. However, the ability of LLMs to generate correct assertions is largely dependent on the input prompt [41]. LLMs can generate security assertions to detect vulnerabilities belonging to various common weakness enumeration (CWE) [45] types using the hardware design specification and description of the hardware CWEs [40].

D. Fixing Code Using LLMs

LLMs have been utilized to not only detect code vulnerabilities but also generate the corrected (vulnerability repaired) code. One such application is through a zero-shot approach, in which prompts are directly engineered to result in an ideal response. This can be implemented on vulnerable software and hardware programs through constructing a prompt that contains the original vulnerable code, some comments describing the vulnerability, and an instruction to generate a fixed version [30]. Although this process resulted in a high success rate in tested synthetic and handcrafted buggy codes, the reliability of the tested LLMs (OpenAI's Codex and others) was determined insufficient to replace automatic program repairs in real-world scenarios [30].

A method to more effectively utilize LLMs for fixing code is fine-tuning. This process updates the weights of a pre-trained language model through training on a dataset related to an intended task, such as vulnerability repair. In particular, researchers used a training dataset of vulnerable C codes [46]. After fine-tuning various LLMs (Llama and Minstral) on this dataset, the test found a significant increase in the accuracy and adaptability of the fine-tuned LLMs' ability to repair vulnerabilities when compared to previous repair techniques [46].

E. Detecting Hardware Trojans Using LLMs

As mentioned in Sec. I, existing techniques for detecting HTs are either ineffective or require extensive training processes and an in-depth understanding of the problem. Off-the-shelf LLMs can aid in overcoming these limitations, as there is no need for additional training or extensive formulations. Researchers have performed a preliminary investigation into the capabilities of LLMs for detecting HTs inserted in the AES benchmark [27]. The investigation reveals that when no context about HT-insertion methods is provided, GPT-3.5 does not detect any HTs, but GPT-4 detects an impressive 81.01% of the HTs. On the other hand, when context about potential HT insertion methods is provided, the performances of GPT-3.5 and GPT-4 jump to 16.36% and 91.67%. GPT-4's performance demonstrates the tremendous power and advantage of LLMs in zero-shot detection of HTs as opposed to traditional methods, which require extensive formulations, coding, and training (for ML-based techniques). Finally, note that the research on LLM-based HT detection is still in its infancy, with many unanswered questions and scope for future research. For instance, [27] only tests on HT-inserted designs, so an understanding of the LLMs' false-positive rates is missing. Moreover, locating HTs in a complex design is another challenging task on which LLMs can be tested.

F. Analyzing Fault Injection Vulnerability Using LLMs

Researchers have also used LLMs in the context of fault injection attacks. In particular, [27] tested the ability of LLMs to compute the fault injection feasibility in finite-state machine transitions. They showed that, when using a sequential multi-step reasoning process, GPT-4 can correctly compute the fault injection feasibility.

III. Future Research Opportunities

LLMs have revolutionized natural language processing, and researchers have explored their potential for various hardware security applications. However, this also introduces security and privacy challenges.

Data Leakage Vulnerabilities. LLMs are trained on vast datasets sourced from the internet and carry inherent risks related to data leakage and misuse. For instance, LLMs can inadvertently memorize and regurgitate sensitive information seen during training, potentially leading to privacy breaches. [47] details this and demonstrates the capability of attackers to extract specific pieces of information from a model, thereby highlighting the risk of sensitive data leakage.

Malicious Content Generation Attacks leverage LLMs to produce harmful or deceptive content, exploiting the models' ability to generate coherent and contextually relevant text. Sophisticated attackers can utilize LLMs to generate or refine malicious code and software exploits [48]. By providing a context or a set of requirements, LLMs can produce code snippets or entire programs designed to breach security systems, exploit vulnerabilities, or perform unauthorized actions on targeted systems.

IV. Conclusion

The advent and integration of LLMs into the semiconductor industry signify a revolutionary shift in the way hardware design and verification processes are approached. LLMs have not only streamlined these processes by automating complex tasks such as translating design specifications into source code and generating robust test cases, but have also enhanced the overall efficiency and reliability of hardware systems. However, as we embrace these advancements, the security implications introduced by the deployment of LLMs within this domain cannot be overlooked. The dual-edged nature of LLMs, capable of both augmenting the design process and potentially introducing vulnerabilities, calls for a balanced approach towards their adoption. It necessitates ongoing research to understand and mitigate the security risks associated with LLMs. As we move forward, the development of advanced security detection and mitigation techniques specifically tailored to counter LLM-induced vulnerabilities will be paramount. Furthermore, the hardware design and semiconductor industry must foster a culture of security-aware LLM usage, ensuring that the benefits of these models are harnessed without compromising the integrity and security of hardware systems. Ultimately, the journey towards fully realizing the potential of LLMs in hardware security is just beginning. With concerted efforts from researchers, industry professionals, and security experts, the future holds the promise of not only innovative but also secure hardware design methodologies, safeguarding the foundational elements of our increasingly digital world.

Acknowledgment

Our research work was partially funded by Intel's Scalable Assurance Program, the US Office of Naval Research (ONR Award #N00014-18-1-2058), and the Lockheed Martin Corporation. This work does not in any way constitute an Intel endorsement of a product or supplier. Any opinions, findings, conclusions, or recommendations expressed herein are those of the authors and do not necessarily reflect those of Intel, the US Government, or the Lockheed Martin Corporation.

References

[1] Y. Naveh, M. Rimon, I. Jaeger, Y. Katz, M. Vinov, E. s Marcu, and G. Shurek, "Constraint-Based Random Stimuli Generation for Hardware Verification," *AI magazine*, vol. 28, no. 3, pp. 13–13, 2007.

[2] S. Whitman, "Virtuosity: Read mode done right," Oct 2017. [Online]. Available: https://community.cadence.com/cadence_blogs_8/b/cic/posts/virtuosity-read-mode-done-right

[3] "VC Formal," https://www.synopsys.com/verification/static-and-formal-verification/vc-formal.html, 2022, Last accessed on 02/11/2022.

[4] "Jasper RTL Apps," https://www.cadence.com/en_US/home/tools/system-design-and-verification/formal-and-static-verification/jasper-gold-verification-platform.html, 2022, Last accessed on 02/11/2022.

[5] Y. Kukimoto, "Introduction to Formal Verification," https://ptolemy.berkeley.edu/projects/embedded/research/vis/doc/VisUser/vis_user/node4.html, 2011, last accessed on 09/12/2022.

[6] G. Dessouky, D. Gens, P. Haney, G. Persyn, A. Kanuparthi, H. Khattri, J. M. Fung, A.-R. Sadeghi, and J. Rajendran, "Hardfails: Insights into Software-Exploitable Hardware Bugs," *USENIX Security Symposium*, pp. 213–230, 2019.

[7] W. Chen, S. Ray, J. Bhadra, M. Abadir, and L.-C. Wang, "Challenges and Trends in Modern SoC Design Verification," *IEEE D&T*, vol. 34, no. 5, pp. 7–22, 2017.

[8] R. Kande, A. Crump, G. Persyn, P. Jauernig, A.-R. Sadeghi, A. Tyagi, and J. Rajendran, "TheHuzz: Instruction Fuzzing of Processors Using Golden-Reference Models for Finding Software-Exploitable Vulnerabilities," *USENIX Security Symposium*, pp. 3219–3236, 2022.

[9] C. Chen, R. Kande, N. Nguyen, F. Andersen, A. Tyagi, A.-R. Sadeghi, and J. Rajendran, "HyPFuzz: Formal-Assisted Processor Fuzzing," *USENIX Security Symposium*, pp. 1361–1378, Aug. 2023. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/chen-chen

[10] J. Hur, S. Song, D. Kwon, E. Baek, J. Kim, and B. Lee, "DIFUZZRTL: Differential Fuzz Testing to Find CPU Bugs," *IEEE Symposium on Security and Privacy*, pp. 1286–1303, 2021.

[11] J. Xu, Y. Liu, S. He, H. Lin, Y. Zhou, and C. Wang, "{MorFuzz}: Fuzzing processor via runtime instruction morphing enhanced synchronizable co-simulation," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 1307–1324.

[12] T. Trippel, K. G. Shin, A. Chernyakhovsky, G. Kelly, D. Rizzo, and M. Hicks, "Fuzzing Hardware Like Software," *USENIX Security Symposium*, 2022.

[13] C. Chen, V. Gohil, R. Kande, A.-R. Sadeghi, and J. Rajendran, "PSOFuzz: Fuzzing Processors with Particle Swarm Optimization," *arXiv preprint arXiv:2307.14480*, 2023.

[14] V. Gohil, R. Kande, C. Chen, A.-R. Sadeghi, and J. Rajendran, "Mabfuzz: Multi-armed bandit algorithms for fuzzing processors," *arXiv preprint arXiv:2311.14594*, 2023.

[15] P. Borkar, C. Chen, M. Rostami, N. Singh, R. Kande, A.-R. Sadeghi, C. Rebeiro, and J. Rajendran, "Whisperfuzz: White-box fuzzing for detecting and locating timing vulnerabilities in processors," *arXiv preprint arXiv:2402.03704*, 2024.

[16] S. Adee, "The hunt for the kill switch," *IEEE Spectrum*, vol. 45, no. 5, pp. 34–39, 2008.

[17] K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss, and F. Piessens, "Plundervolt: Software-based fault injection attacks against Intel SGX," in *IEEE S&P*, 2020, pp. 1466–1482.

[18] H. Chen, X. Zhang, K. Huang, and F. Koushanfar, "AdaTest: Reinforcement learning and adaptive sampling for on-chip hardware Trojan detection," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 2, pp. 1–23, 2023.

[19] V. Gohil, S. Patnaik, H. Guo, D. Kalathil, and J. Rajendran, "DETERRENT: Detecting Trojans using Reinforcement Learning," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 697–702.

[20] A. Sarihi, P. Jamieson, A. Patooghy, and A.-H. A. Badawy, "Multi-criteria hardware trojan detection: A reinforcement learning approach," in *2023 IEEE 66th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 2023, pp. 1093–1097.

[21] V. Gohil, H. Guo, S. Patnaik, and J. Rajendran, "ATTRITION: Attacking Static Hardware Trojan Detection Techniques Using Reinforcement Learning," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 1275–1289.

[22] H. Guo, S. Saha, V. Gohil, S. Patnaik, D. Mukhopadhyay, and J. J. Rajendran, "ExploreFault: Identifying Exploitable Fault Models in Block Ciphers with Reinforcement Learning," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[23] V. Gohil, S. Patnaik, D. Kalathil, and J. Rajendran, "Attackgnn: Redteaming gnns in hardware security using reinforcement learning," *arXiv preprint arXiv:2402.13946*, 2024.

[24] T. Kocmi and C. Federmann, "Large language models are state-of-the-art evaluators of translation quality," 2023.

[25] W. Zhang, Y. Deng, B. Liu, S. J. Pan, and L. Bing, "Sentiment analysis in the era of large language models: A reality check," 2023.

[26] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang, "Large language models for software engineering: Survey and open problems," 2023.

[27] D. Saha, S. Tarek, K. Yahyaei, S. K. Saha, J. Zhou, M. Tehranipoor, and F. Farahmandi, "Llm for soc security: A paradigm shift," *arXiv preprint arXiv:2310.06046*, 2023.

[28] S. Liu, W. Fang, Y. Lu, Q. Zhang, H. Zhang, and Z. Xie, "Rtlcoder: Outperforming gpt-3.5 in design rtl generation with our open-source dataset and lightweight solution," *arXiv preprint arXiv:2312.08617*, 2023.

[29] R. Zhong, X. Du, S. Kai, Z. Tang, S. Xu, H.-L. Zhen, J. Hao, Q. Xu, M. Yuan, and J. Yan, "Llm4eda: Emerging progress in large language models for electronic design automation," *arXiv preprint arXiv:2401.12224*, 2023.

[30] H. Pearce, B. Tan, B. Ahmad, R. Karri, and B. Dolan-Gavitt, "Examining Zero-Shot Vulnerability Repair with Large Language Models," 2022.

[31] M. DeLorenzo, A. B. Chowdhury, V. Gohil, S. Thakur, R. Karri, S. Garg, and J. Rajendran, "Make every move count: Llm-based high-quality rtl code generation using mcts," *arXiv preprint arXiv:2402.03289*, 2024.

[32] K. Chang, H. Ren, M. Wang, S. Liang, Y. Han, H. Li, X. Li, and Y. Wang, "Improving large language model hardware generating quality through post-llm search."

[33] Synopsys, "Unleash the Power of AI at Synopsys," https://www.synopsys.com/ai.html, Last accessed on 01/03/2024.

[34] X. Zhou, T. Zhang, and D. Lo, "Large language model for vulnerability detection: Emerging results and future directions," 2024.

[35] W. Fu, K. Yang, R. G. Dutta, X. Guo, and G. Qu, "Llm4sechw: Leveraging domain-specific large language model for hardware debugging," in *2023 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, 2023, pp. 1–6.

[36] S. Ullah, M. Han, S. Pujar, H. Pearce, A. Coskun, and G. Stringhini, "Can large language models identify and reason about security vulnerabilities? not yet," 2023.

[37] M. Li, W. Fang, Q. Zhang, and Z. Xie, "Specllm: Exploring generation and review of vlsi design specification with large language model," 2024.

[38] J. Hu, Q. Zhang, and H. Yin, "Augmenting greybox fuzzing with generative ai," 2023.

[39] H. Witharana, Y. Lyu, S. Charles, and P. Mishra, "A survey on assertion-based hardware verification," *ACM Comput. Surv.*, vol. 54, no. 11s, sep 2022. [Online]. Available: https://doi.org/10.1145/3510578

[40] S. Paria, A. Dasgupta, and S. Bhunia, "Divas: An llm-based end-to-end framework for soc security analysis and policy-based protection," *arXiv preprint arXiv:2308.06932*, 2023.

[41] R. Kande, H. Pearce, B. Tan, B. Dolan-Gavitt, S. Thakur, R. Karri, and J. Rajendran, "Llm-assisted generation of hardware assertions," *arXiv preprint arXiv:2306.14027*, 2023.

[42] M. Orenes-Vera, M. Martonosi, and D. Wentzlaff, "Using llms to facilitate formal verification of rtl," *arXiv e-prints*, pp. arXiv–2309, 2023.

[43] W. Fang, M. Li, M. Li, Z. Yan, S. Liu, H. Zhang, and Z. Xie, "Assertllm: Generating and evaluating hardware verification assertions from design specifications via multi-llms," *arXiv preprint arXiv:2402.00386*, 2024.

[44] M. Hassan, S. Ahmadi-Pour, K. Qayyum, C. K. Jha, and R. Drechsler, "Llm-guided formal verification coupled with mutation testing."

[45] MITRE, "Hardware Design CWEs," https://cwe.mitre.org/data/definitions/1194.html, 2019, Last accessed on 04/08/2021.

[46] D. de Fitero-Dominguez, E. Garcia-Lopez, A. Garcia-Cabot, and J.-J. Martinez-Herraiz, "Enhanced automated code vulnerability repair using large language models," 2024.

[47] N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson *et al.*, "Extracting training data from large language models," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2633–2650.

[48] P. V. S. Charan, H. Chunduri, P. M. Anand, and S. K. Shukla, "From text to mitre techniques: Exploring the malicious use of large language models for generating cyber attack payloads," 2023.