Resilient DevSecOps: Leveraging Large Language Models and Chaos Engineering for Automated Threat Hypothesis Validation

Author
Miguel Santiago Betancourt Alonso

Work presented as a requirement for the degree of
Master in Applied Mathematics and Computer Science

Director
PhD. Daniel Orlando Díaz López

School of Science and Engineering
Master in Applied Mathematics and Computer Science
University del Rosario

Bogotá - Colombia
2026

Acknowledgment

The author would like to express sincere gratitude to the research team with whom the first version of the IJIS 2025 paper was developed and presented. In particular, special thanks are extended to Professor *Daniel Díaz* from University of Rosario and *Mario Marín* from the University of Murcia for their guidance, collaboration, and valuable discussions throughout the research process. This collaboration represented a new and enriching learning experience that significantly contributed to the development of this work.

Abstract

The securitization of the software development lifecycle is a practice that enables companies to produce code that meets the three fundamental pillars of security: integrity, confidentiality, and availability of the processed data, as well as the services provided within their production applications. Currently, it is mandatory to integrate practices from the Secure Software Development Lifecycle (SSDLC) into the team's tasks due to the increasing rise of security threats. Development teams are typically composed of both technical and non-technical personnel who participate in the early stages of the SSDLC, such as planning and design. However, many of these members lack knowledge in cybersecurity. In addition to the lack of knowledge, the integration of securitization tools into the SSDLC is hampered by the fact that these tools are applied manually and require significant time for construction. Furthermore, the lag in addressing new threats results in the final product being vulnerable to cyberattacks due to outdated security components or policies. This work, presented as a degree project for the Master's in MACC, proposes the integration of Large Language Models (LLMs) and the methodology of Security Chaos Engineering (SCE) to facilitate the incorporation of security-focused tasks within the Secure Software Development Life Cycle (SSDLC). On one hand, LLMs automate the tasks of constructing and interpreting attack and defense trees, facilitating the generation of hypotheses about attack scenarios. On the other hand, SCE provides an evaluation of the system's resilience, stability, and recovery capabilities, resulting from the execution of a set of experiments in a controlled DevSecOps environment aimed at exploiting system vulnerabilities.

Contents

0.1 Introduction

The cost of software securitization can increase by up to a factor of 15 during the testing phase compared to the cost of implementing software assurance during the design and requirements analysis stages [1]. This occurs because modifications made early in the development process affect components independently, allowing issues to be addressed in isolation. In contrast, changes introduced in later stages can no longer be applied at the component level; instead, they must be made at the project level, which significantly increases complexity, effort, and cost.

Software securitization in companies is usually carried out, at best, during the development stage. However, the measures applied during this phase are often insufficient and, at times, too late to ensure that the final product is secure. As the development cycle progresses, the software accumulates vulnerabilities, leading to multiple security gaps in the final product, as explained in [2]. Even after addressing the vulnerabilities identified during the development phase, the software may still have security flaws from the design stage.

In many cases, development teams lack personnel trained in secure software development and do not have access to specialized cybersecurity consulting services. According to a survey conducted in [3], 30% of organizations reported struggling with a shortage of cybersecurity skills. Furthermore, the time dedicated to software securitization is generally not considered in the planning and requirements analysis stages.

Small and medium-sized enterprises (SMEs) are the most affected by cyberattacks, and in some countries, they represent 90% of all businesses, contributing up to a third of the GDP [4]. While software securitization issues affect large, medium, and small companies, SMEs are most often the ones that lack a specialized team or department, budget, and time allocated for cybersecurity. This makes their software products highly vulnerable to attacks. Despite their size, these SMEs companies handle sensitive information both from their clients and their own business, which must be adequately protected.

Today, more and more companies are choosing to migrate their systems to the cloud. However, migrating infrastructure and software development to the cloud does not ensure the implementation of an Secure Software Development Lifecycle (SSDLC) in development projects. According to the National Institute of Standards and Technology (NIST), the cloud consists of three service models: IaaS, PaaS, and SaaS. Each model has its own infrastructure characteristics and configuration freedoms, with shared responsibilities, different data storage methods, and access protocols [5]. The level of user involvement in software securitization depends on the service model contracted. However, the physical securitization of the cloud is the provider's responsibility, while the securitization of software within the cloud is the user's responsibility in the IaaS and PaaS service models, who must apply secure development principles to protect their applications.

Failing to implement a SSDLC not only leads to security issues but also results in financial and reputation consequences in the market. In 2014, Yahoo's data breaches exposed the confidentiality of information from 3 billion user accounts, which led to a $350 million reduction in Verizon's purchase offer for the platform [6]. This is just one of many examples that illustrate the magnitude of the reputation and financial consequences that arise when the final software product is insecure.

Given the above, it is clear that for companies (regardless of their size), implementing an SSDLC in their software development projects is an urgent necessity, one that should begin in the early phases of the SSDLC. The main challenge lies in the lack of knowledge regarding secure development concepts and practices within the areas involved in software creation (which are not limited solely to technical personnel), as well as the economic constraints faced by SMEs. Implementing an SSDLC provides security, economic, and reputational benefits for companies that develop software. Having a tool that automates part of the software securitization task makes it easier to protect the integrity, confidentiality, and availability of

the data processed in projects carried out within a DevSecOps environment.

0.2 Related Works

This work builds upon two complementary research areas that are central to the study and experimentation of resilience in DevSecOps environments: Artificial Intelligence (AI) and the Security Chaos Engineering (SCE) paradigm. The following sections review the most relevant proposals in these domains, examining their objectives, methodologies, and contributions, and highlighting how they collectively advance the state of the art in DevSecOps security assurance. In particular, Section 0.2.1 reviews the main advances in the use of AI—specifically Large Language Models (LLMs)—for securing software development, while Section 0.2.2 surveys recent contributions in the area of SCE.

0.2.1 Securing Development with Artificial Intelligence

Artificial Intelligence, and particularly Large Language Models (LLMs), has gained increasing relevance as a means to enhance secure development workflows. Traditional security tools such as Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), or Infrastructure-as-Code (IaC) analyzers typically rely on predefined rules and exhibit limited capacity for semantic reasoning or cross-layer analysis. In contrast, LLMs provide contextual understanding, natural-language processing capabilities, and flexible inference mechanisms, thereby enabling new opportunities to automate and strengthen security assurance activities throughout the DevSecOps pipeline [7].

In this context, in [8] assessed the ability of GPT-based models to detect insecure programming patterns. Their results show that LLMs can identify subtle code-level weaknesses and articulate coherent explanations, effectively complementing conventional static analysis tools.

Similarly, the authors in [9], analyzed the use of generative models for vulnerability identification during code review. Their work demonstrated that LLMs can highlight suspicious constructs and provide actionable remediation suggestions, supporting developers in addressing vulnerabilities earlier in the software development lifecycle.

McIntosh in [10] explored the integration of LLMs into continuous integration pipelines for automated pull-request review. Their evaluation showed that LLMs can reduce manual inspection effort by producing relevant security insights with acceptable precision.

Moreover, authors in [11] investigated the ability of LLMs to classify and explain common weakness patterns across diverse codebases. Their findings indicate that such models generalize across programming languages and frameworks, enabling more uniform vulnerability classification in heterogeneous environments.

Beyond source-code analysis, Cankar in [12] proposed machine-learning mechanisms for analyzing IaC configurations. Their approach detects misconfigurations and risky deployments by learning from IaC patterns, surpassing the capabilities of traditional rule-based linters.

Beyond secure development tasks, LLMs have also been applied to model attacker behavior, generate adversarial tactics, and anticipate multi-step attack strategies. These efforts shift the focus from code-level reasoning toward understanding how attackers plan, adapt, and execute offensive operations—an aspect that is particularly relevant for threat modeling and security analysis.

In this direction, Gadyatskaya in [13] evaluated the ability of LLMs to generate attack trees from textual scenario descriptions. Their results confirm that LLMs can produce structurally coherent trees, while also highlighting the need for expert supervision and formal structural constraints to mitigate inaccuracies.

Similarly, Zhang in [14] studied the use of LLMs for strategic cyber reasoning, showing that models can articulate multi-step adversarial plans and evaluate attacker strategies at an abstract level.

Diaf in [15] proposed a learning-based mechanism to predict zero-day cyberattacks in IoT environments. Their approach integrates contextual factors to anticipate exploitation attempts, demonstrating the value of AI-driven forecasting for proactive defense.

Furthermore, Zhang in [16] explored the generation of adversarial Tactics, Techniques, and Procedures (TTPs) using LLM prompts. Their results indicate that LLMs can transform natural-language descriptions into structured adversarial techniques inspired by the MITRE ATT&CK framework.

Wang in [17] presented a framework for simulating full attack chains by combining LLM-organized knowledge with structured reasoning. Their system generates multi-step attack narratives along with potential mitigations, illustrating how LLMs can support scenario-based threat modeling.

Additionally, Haryanwo in [18] examined how LLMs can contextualize cybersecurity events to assist analysts during incident triage. Their assessment highlights both the potential benefits and the limitations related to ambiguous or inconsistent model outputs.

Finally, authors in [19] investigated the use of LLM-generated adversarial behaviors to train autonomous cyber-defense agents. Their work demonstrates the value of LLMs in producing realistic attack patterns that enrich training datasets for defensive reinforcement-learning systems.

Collectively, these works illustrate the breadth of AI-driven research in secure development and adversary modeling. While significant progress has been made across code analysis, IaC security, threat simulation, and attacker-behavior prediction, recurring challenges remain, including hallucination, scalability limitations, and the lack of formal structural guarantees. These limitations motivate the integration of LLM-based reasoning with formal security models and empirical validation mechanisms.

0.2.2 SCE-Powered Proposals

Security Chaos Engineering (SCE) extends the principles of Chaos Engineering to the validation of security controls under realistic adversarial perturbations. Unlike traditional security testing, which often focuses on predefined scenarios or static configurations, SCE introduces controlled disruptions to assess whether defensive mechanisms behave as expected. Existing works apply SCE across cloud environments, cyber–physical systems, distributed infrastructures, and DevSecOps pipelines, illustrating the growing versatility of this paradigm while also exposing methodological inconsistencies and limited automation.

One of the earliest security-focused chaos experimentation frameworks is *ChaoSlingr* [20], which enables controlled fault injection in cloud environments, focusing on access-control misconfigurations and monitoring failures. While the framework demonstrates the value of security-oriented perturbations, its experiments remain largely handcrafted and dependent on operator expertise.

Similarly, *CloudStrike* [21] introduced a structured method for conducting security chaos experiments in cloud-native infrastructures, with an emphasis on validating logging pipelines, identity-management controls, and configuration integrity. The work illustrates how SCE can uncover fragile dependencies hidden beneath apparently resilient architectures.

SCE has also been applied to cyber–physical systems (CPS). Charalambous in [22] demonstrated that inducing controlled disruptions in CPS environments can reveal safety–security interactions and operator errors that traditional reliability testing may overlook.

Sharieh in [23] investigated the application of SCE to API ecosystems, showing that security perturbations can expose weaknesses in authorization flows, error-handling routines, and dependency chains, thereby highlighting the need for continuous experimentation in highly distributed systems.

Additionally, Bailey in [24] explored SCE in complex system-of-systems contexts, revealing emergent vulnerabilities arising from interactions among heterogeneous components and

underscoring the importance of system-level reasoning.

The SCENE guidelines proposed by Jolak in [25] provide a comprehensive systematization of existing SCE research. Through an extensive literature review, they identify recurring methodological patterns, gaps in current experimentation practices, and the need for unified taxonomies of perturbations and evaluation metrics.

From a practitioner perspective, Elumalai [26] discussed the integration of chaos engineering techniques into DevSecOps pipelines, demonstrating how security chaos experiments can validate defensive assumptions, detect configuration drift, and support compliance in agile environments.

A major advancement toward automation in SCE is represented by *ChaosXploit* [27], which integrates attack-tree knowledge with an experiment execution engine to automatically derive hypotheses, evaluate defenses, and manage rollback procedures. Follow-up work by Bedoya in[28, 29] extended this framework by incorporating natural-language processing and LLMs to extract threat information from user stories, bridging design-time threat modeling with runtime experimentation.

Overall, these works demonstrate the evolution of SCE from manual, scenario-driven experimentation toward more systematic and automated approaches. However, most existing proposals still rely on handcrafted attack scenarios and lack mechanisms for dynamic adaptation. These limitations motivate the integration of AI-generated attack and defense trees into SCE workflows, enabling more scalable, data-driven, and continuously validated security experimentation.

0.3 Objectives

General Objective

To develop an innovative methodological workflow for evaluating the resilience of information systems within DevSecOps environments, integrating the identification and modeling of attack vectors and scenarios generated semi-automatically using Large Language Models (LLMs), and their experimental validation through the design and execution of Security Chaos Engineering (SCE)-based experiments.

Specific objectives

- OBJ1: Validate a previous proposal (papers [28] and [29]) that integrated SCE, into DevSecOps practices, and improve it using both general and specialized LLMs to create an attack-defense tree.
- OBJ2: Generate attack-defense trees with the support of a LLM, guaranteeing that it contains information usable to implement attack procedures and countermeasures.
- OBJ3: Generate SCE experiments from the previously generated attack-defense tree, that allow to define hypotheses, observability conditions, and a scenario where to deploy the experiment.
- OBJ4: Integrate the findings into a Secure Software Development Lifecycle (SSDLC), ensuring that hypothesis formulation and security validations are integrated within a DevSecOps environment.

0.4 Methodology

This work is framed within an experimental and applied approach aimed at enhancing the security and resilience of information systems developed under a Secure Software Development Life Cycle (SSDLC). The research objective is to evaluate the use of Large Language Models (LLMs) integrated into a methodological workflow that enables the semi-automatic generation of attack and defense trees, as well as the construction of Security Chaos Engineering (SCE) experiments. Additionally, a comparative analysis is conducted among widely used general-purpose LLMs to determine their performance in threat modeling tasks, specifically in the structured generation of attack and defense trees and in the definition of SCE experimental scenarios.

0.4.1 Research Methodology

To achieve the objectives proposed in this work, a set of phases was defined and organized as sequential tasks within the graph-based methodology illustrated in Figure 1. This methodology structures the research process into well-defined stages, ensuring traceability and reproducibility of the experimental workflow.

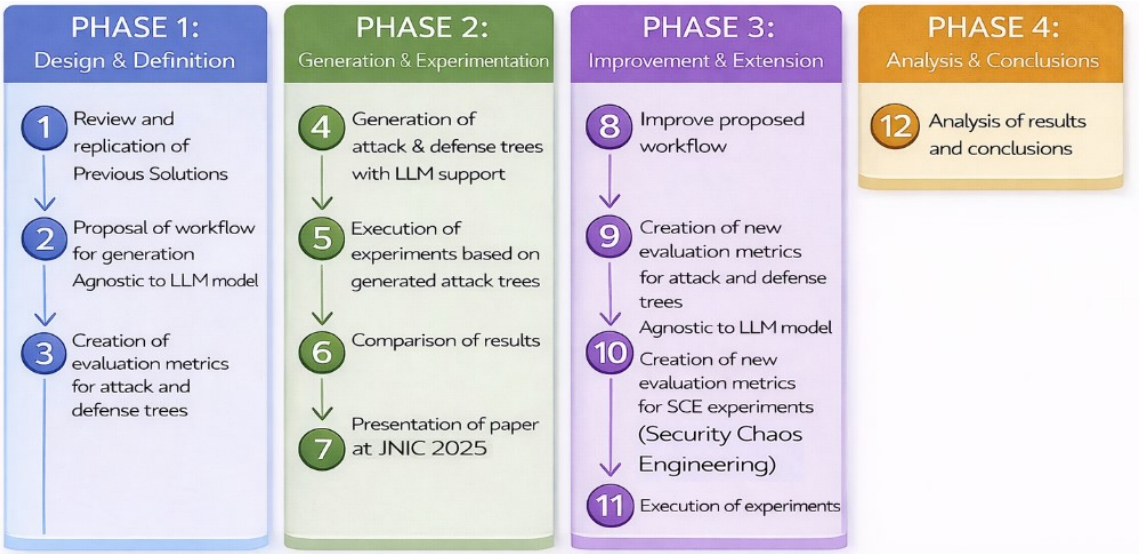

Figure 1: Research Methodology organized by phases

0.4.2 Phase 1: Replication of Prior Work

As part of the initial phase, a replication of prior research was conducted to validate and extend existing approaches related to the use of artificial intelligence in Security Chaos Engineering. Specifically, the studies presented at the ARES conference [28] and in the International Journal of Information Security (IJIS) [29] were replicated.

In particular, the article “Securing Cloud-Based Military Systems with Security Chaos Engineering and Artificial Intelligence”, presented at ARES by Martín Bedoya, motivated the replication of the task of processing user stories from a military supply system using an NER model [30]. This model allows security analysts or software developers to automatically extract entities and actions from early stages of the Secure SDLC (Secure Software Development Life Cycle), such as the design phase. This information is relevant for describing the functional behavior of the system.

The implementation of the NER model made it possible to become familiar with natural language processing techniques and to use its results as input for the manual construction

of the risk model of the military supply chain system. Likewise, the replication of the article [29] enabled progress toward the construction of attack and defense trees. In this work, presented at IJIS by Martín Bedoya, an integration flow is proposed that combines the use of large language models (LLMs) with system context to generate such trees as input for the manually construction of Security Chaos Engineering (SCE) experiments.

To replicate this second article, the context of the military supply system, the cloud components, and the attack objectives defined in the original work were used as input (prompt context) for the LLM. An exploratory process was conducted with various language models available online, such as ChatGPT-4 [31], Qwen [32], and DeepSeek [33], for generating attack and defense trees and transforming them into SCE experiments, which enabled the proposed attacks to be executed in a controlled environment. The results of the ARES article replication are presented in Section 0.6, and those of the IJIS article replication in Section 0.7 of the Results. The work carried out up to this point made it possible to fulfill OBJ1 of this project.

0.4.3 Phase 1: LLM-Based Workflow for Attack and Defense Tree Generation

The proposed workflow, based on the use of Large Language Models (LLMs), is structured into three sequential prompting blocks in its initial version, which is illustrated in Subsection 0.8.1. Each block fulfills a specific function within the automated process aimed at generating a validated attack–defense tree.

The first set of blocks corresponds to the contextualization phase, in which the security analyst provides the model with detailed information about the simulated military supply chain, the AWS components that constitute the system architecture (e.g., CodeBuild, CodePipeline, EC2, and IAM), and the predefined attack objectives. This stage establishes the operational context and defines the boundaries of the system to be analyzed.

The second set of block focuses on the definition of quality metrics for attack and defense trees, together with the structured identification of attack vectors associated with the specified objectives. Based on this input, the LLM generates an initial version of the attack–defense tree, which represents the main output of this phase.

The third set of blocks concentrates on structural and representational refinement of the generated tree, enabling improvements in clarity, hierarchical organization, and formal consistency. As a result of this block, a validated and well-structured attack–defense tree is obtained, constituting the final outcome of the initial version of the workflow.

0.4.4 Phase 1: System Context

The problem addressed is developed within a controlled environment that simulates a real DevOps setting deployed on Amazon Web Services (AWS). Within this environment, and based on the designed methodological workflow, Large Language Models (LLMs) are used to semi-automatically generate attack and defense trees. Based on these generated attack–defense trees, a security analyst defines and executes the corresponding Security Chaos Engineering (SCE) experiments. These experiments are subsequently executed in the controlled environment in order to observe their impact on the security and resilience of the system.

The study seeks to evaluate the performance of different LLMs in the structured generation of threat models and experimental scenarios through the design and implementation of specific quality metrics. The results obtained from these metrics enable an objective comparison among the evaluated models and allow assessing their potential contribution to strengthening information systems developed under an SSDLC through the incorporation of the proposed workflow.

The controlled AWS environment, designed to simulate a DevOps application associated with a military-sector supply chain, constitutes the experimental scenario in which the proposed workflow is executed. The implemented architecture integrates services such as AWS CodeBuild, CodePipeline, EC2 instances, and Identity and Access Management (IAM), among other components commonly present in continuous integration and deployment environments. These services were selected due to the existence of previously documented attack vectors in specialized repositories such as HackTricks and used in the prior works Subsection 0.4.2, which enables the modeling of realistic and technically grounded threat scenarios.

The simulated supply chain is deployed under a hybrid cloud architecture, which introduces both internal and external attack surfaces. In this context, the system may be compromised by privileged internal actors (insiders) or external attackers. The military domain was selected to maintain coherence with prior research on the topic and due to the strategic criticality of this sector, where the implementation of active defense mechanisms and operational resilience is particularly relevant.

The architecture of the military supply chain application deployed in the AWS cloud is shown in Figure 2. The security analyst is responsible for designing and executing security tests and for assessing the resilience of the system. In this role, the analyst must reason and act as an adversary, adopting an attacker's mindset in order to realistically evaluate potential weaknesses.

Specifically, the security analyst is in charge of creating controlled experimental conditions in which system vulnerabilities can be exploited. These vulnerabilities may include, but are not limited to, misconfigured access control policies, excessive permissions granted to users or roles, exposed or hard-coded credentials, inadequate segregation of duties, insecure service integrations, and insufficient monitoring or logging mechanisms. By deliberately recreating such conditions, the analyst enables the systematic evaluation of defensive controls and the overall resilience of the cloud-based system.

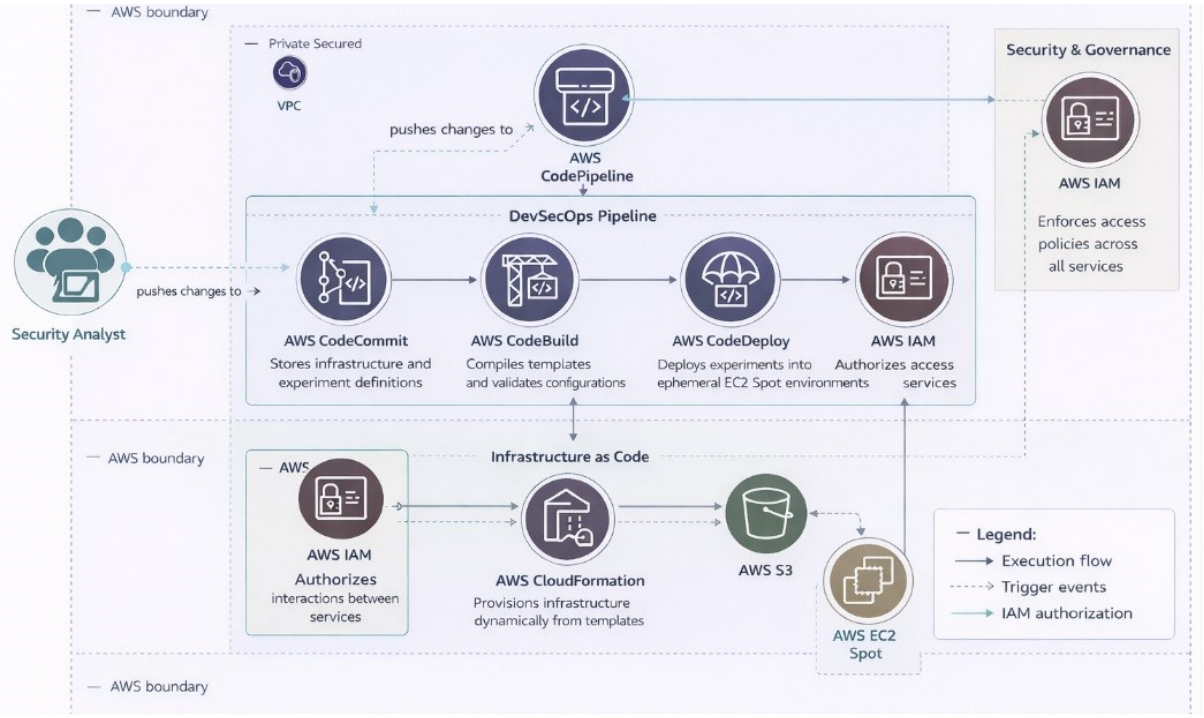

Figure 2: Proposed Experimentation Architecture

0.4.5 Phase 1: Quality Metrics for Attack and Defense Trees

Unlike previous works in this area, the proposed workflow incorporates an explicit quality validation stage based on a set of defined metrics. The objective of this stage is to obtain attack and defense trees with a well-defined hierarchy and clear, complete content at each node, thereby facilitating the accurate interpretation, reproducibility of attack paths, and validation of defensive logic.

The metrics designed to evaluate the quality of the attack and defense trees generated within the *initial version of the proposed workflow* are presented in Subsection 0.8.2. This first version includes *three* quality metrics, which enable the objective quantification of both the structural properties and the semantic content of the generated trees, supporting automated decision-making during the tree generation and refinement process.

The definition of these metrics is grounded in formal principles of attack tree modeling [34] and [35], as well as in the characterization of tactics and techniques described in the MITRE ATT&CK framework [36], ensuring alignment with widely recognized cybersecurity standards.

0.4.6 Phase 2: Attack and Defense Trees Generation

With the metrics defined for validating the quality of attack and defense trees, and with the inclusion of a refinement step for the attack tree within the workflow, the generation of these trees was carried out using ChatGPT-4 and Qwen. The tree obtained with ChatGPT-4 and Qwen (a general-purpose LLMs) is shown in Figure 8 and 7. These results enabled an initial quality comparison between the tree generated by ChatGPT-4 and the one generated by Qwen, as presented in Subsection 0.8.4.

0.4.7 Phase 2: SCE Experiment Construction And Execution

Subsequently, the construction and execution of the SCE experiment described in the selected attack and defense tree were performed using the Chaostoolkit [37] framework together with the Chaosaws [38] Python extension integrated in ChaosXploit[39]. The results of the experiment execution are explained in Subsection 0.8.5. Once it was verified that the step-by-step procedure described in the tree enabled the reproduction of the attack and the generation of the preconfigured alerts, fulfill OBJ2 of this thesis was considered achieved.

0.4.8 Phase 2: Presentation Of Paper At JNIC 2025

As a result of the consolidated work carried out during this research, a paper was presented at the National Cybersecurity Research Conference (JNIC 2025), held in Zaragoza, Spain, entitled *"Anticipating Adversary Behavior in DevSecOps Scenarios through Large Language Models"* [40, 41]. This article presented the results achieved in the generation of attack trees and the construction and execution of Security Chaos Engineering (SCE) experiments, highlighting the proposed quality validation metrics for attack and defense trees, the refined workflow that incorporates these metrics, and the comparative analysis of different LLMs for the task of attack and defense tree generation. The full paper is included in the annexes of this document for reference (JNIC 2025).

0.4.9 Phase 3: Improve Proposed Workflow

In order to improve the proposed workflow, the objective was to build upon the work developed for the paper presented at JNIC 2025 by incorporating enhancements derived from that initial deliverable. In particular, the focus was placed on strengthening the LLM-based workflow for attack and defense tree generation. This led to the definition of four additional

quality validation metrics, which were incorporated into the three metrics originally proposed in the initial version of the workflow. The inclusion of these new metrics was motivated by limitations observed in earlier iterations, where some generated tree nodes exhibited ambiguous or incomplete attack or defense descriptions, and in certain cases contained empty input fields. As a result, a total of seven quality validation metrics for attack and defense trees were defined. The complete set of proposed metrics is presented in Subsection 0.9.3.

Beyond the incorporation of quality validation metrics for the generated attack and defense trees, the extended workflow aims to maximize the level of automation of the overall process. To this end, an additional task corresponding to the generation of Security Chaos Engineering (SCE) experiments was introduced as a new block within the workflow. This block includes a refinement step implemented as an iterative loop, whose continuation condition is defined by the fulfillment of a target quality threshold derived from a set of SCE experiment quality metrics. Consequently, a new set of quality metrics specifically designed for evaluating SCE experiments was also proposed, enabling systematic validation and controlled execution of the generated experiments.

0.4.10 Phase 3: Creation of New Evaluation Metrics for Attack and Defense Trees

The new evaluation metrics for attack and defense tree that are shown in Table 5 were designed with the objective of producing tree nodes that contain truthful, complete, and sufficient information to enable the reproduction of each attack or defense step. The results obtained from the generated attack and defense trees indicate that these metrics help limit or prevent hallucinations produced by LLMs, while also promoting the abstraction and generalization of attack and defense patterns.

Figure 13 presents an example branch of an attack and defense tree generated using the proposed evaluation metrics with the selected LLM. In addition, the application of these quality metrics made it possible to verify that iterative refinement of the prompt context—through a process analogous to backpropagation—progressively improves the structural and semantic quality of the generated trees.

0.4.11 Phase 3: Creation of New Evaluation Metrics for SCE Experiments

To further advance the automation of the proposed workflow, the integration of automatic Security Chaos Engineering (SCE) experiment generation was introduced. This extension required the definition of dedicated validation metrics for LLM-generated SCE experiments, as illustrated in Figure 12. For this phase, Claude Sonnet [42] was selected, as prior experiments demonstrated its ability to produce more technically grounded abstractions of attack vectors and defense implementations. Moreover, it showed a strong capability to translate these abstractions into executable code, making it particularly suitable for automated SCE experiment generation.

With the inclusion of experiment generation as a new block in the extended workflow, a set of quality validation metrics specifically tailored to SCE experiments was defined. These metrics are presented in Subsection 0.9.2 and enable the structured and automated generation of valid and reproducible SCE experiments.

As part of the workflow output, the system produces not only the attack and defense tree represented as a graph, but also the necessary files required for executing the attacks and validating the corresponding defenses. The generated `.json` file defines the experiment structure, including the hypothesis, probes, and actions, as illustrated in Figure 14. Additionally, the generated `.py` files implement the required functions which, supported by frameworks such as *Chaos Toolkit* and *Chaos AWS*, provide the capabilities to connect to the cloud

environment and execute the attack and defense vectors in a controlled and reproducible manner.

Using the generated artifacts, the SCE experiment was executed, and the resulting detailed execution log is shown in Figure 15. The execution enabled the validation of the defined hypothesis through the successful operation of the probes, as well as the execution of the attack vectors and the activation of the configured defenses via the specified actions. The results obtained confirm the successful fulfillment of objectives OBJ3 and OBJ4.

0.4.12 Phase 4: Analysis of Results and Conclusions

The work carried out during the implementation of this phase-structured methodology enabled the progressive generation of concrete deliverables for each phase, as well as the validation of different workflow versions for the generation of attack and defense trees and the construction of Security Chaos Engineering (SCE) experiments. A more detailed analysis of these results is presented in Section 0.10.

The final outcome of this research supports several key conclusions. First, a clear improvement was observed when comparing the initial attack and defense trees generated in the early phases with the final trees produced by the extended workflow, both in terms of structure and content quality. Second, the proposed workflow demonstrates that it is feasible to systematically derive executable SCE experiments directly from attack and defense trees, effectively bridging threat modeling and experimental security validation. Finally, this work also highlights relevant future challenges, particularly the need to achieve full automation of the workflow by minimizing or eliminating human intervention during the experimental phase.

0.5 Results

As a result of the work of my thesis research titled **Resilient DevSecOps: Leveraging Large Language Models and Chaos Engineering for Automated Threat Hypothesis Validation**, the following outcomes were achieved:

- Replication of the work presented at the ARES conference by Martín Bedoya [28], which explores the integration of artificial intelligence tools with the Security Chaos Engineering (SCE) methodology for cyber defense tasks in DevSecOps scenarios.
- Replication of experiments 4 and 5 proposed in the IJIS paper [29]. This included reproducing: (i) the privilege-escalation scenario in which AWS GuardDuty detects the misuse of AWS CodeBuild credentials to gain elevated access (Experiment 4 – CB), validating the corresponding alerts, misconfiguration conditions, and adversarial pathway described in the original work; and (ii) the SQL-injection scenario in which AWS CodeGuru-Security identifies insecure coding practices within the CI pipeline (Experiment 5 – CP), confirming the vulnerability detection events, developer-behavior assumptions, and database-exfiltration attack path documented in the reference experiments.
- Contribution to the paper presented at JNIC 2025 [40, 41], with the following specific achievements:
  - Generation of attack and defense trees through the creation and refinement of a prompt set for language models.
  - Definition and formulation of evaluation metrics to assess the quality and effectiveness of the attack and defense trees generated by large language models (LLMs).
  - Execution of practical experiments to validate the attack paths derived from the generated attack and defense trees.
- Contribution to the extended version of the research paper (Submission and publication of a scientific research paper), which is still under development at the time of presenting this work, with the following specific achievements:
  - Definition and formulation of four new evaluation metrics to assess the quality and effectiveness of the attack and defense trees generated by Large Language Models (LLMs).
  - Definition and formulation of new evaluation metrics to evaluate the quality and effectiveness of the SCE experiments generated by LLMs and derived from the attack–defense tree structure.
  - Proposal of a modified workflow integrating the automatic generation of attack–defense trees and SCE experiments, resulting in a fully automated end-to-end process.

0.6 Replication of Paper Presented at ARES Conference

The paper "Securing Cloud-Based Military Systems with Security Chaos Engineering and Artificial Intelligence" by Martin Bedoya, presented at the ARES Conference, was reviewed. The research proposes a development cycle scenario for a cloud-hosted military system. It employs a Named Entity Recognition (NER) model along with Security Chaos Engineering (SCE) to model and experimentally validate threats during the early stages of software design.

A key contribution highlighted in the paper is the integration of artificial intelligence to support threat modeling, as well as the use of tools such as attack trees. These attack trees enabled the construction of Security Chaos Engineering experiments. Four distinct experiments were proposed following the SCE methodology. These experiments, based on the attack trees, aimed to validate potential vulnerabilities in services deployed on the AWS cloud platform.

0.6.1 Implementation of the NER Model

As a first step toward replicating the work presented in ARES conference, the Named Entity Recognition (NER) model used for processing user stories was implemented. For this purpose, the project was cloned from GitHub at [30]. The model was executed using the example user story provided in the original paper as input.

NER is an entity recognition model that, through natural language processing (NLP), identifies and classifies entities mentioned in a given text. As a result, the NER model identifies the subject and the action described in the user story. For example, in the sentence: "As a logistic officer I would like to report information about consumed, leftover and forecasted resources" the model identifies the subject as "logistic officer" and the action as "report."

In Figure 3, the processing of the test user story is shown using the best model trained on BERT with WordNet Synonym Augmentation. The model successfully identifies *"logistic officer"* as *Data*, *"report"* as *Processing*, and *"consumed"* as *PII* (Personally Identifiable Information).

```
2025-05-31 23:07:15,057 SequenceTagger predicts: Dictionary with 10 tags: <unk>, O, B-Data, B-Process
ing, B-PII, I-PII, I-Processing, I-Data, <START>, <STOP>
Sentence[18]: "As a logistic officer I would like to report information about consumed, leftover and
forecasted resources." → ["logistic officer"/Data, "report"/Processing, "consumed"/PII]
```

Figure 3: User Story Processing with NER Model

0.6.2 Generating System Topology Using IriusRisk

IriusRisk [43] is a threat modeling platform that reports potential risks associated with the incorrect implementation of components represented in a system topology. In an effort to replicate the work presented in the ARES paper, the same topology was recreated and the threat modeling was generated using the tool.

The proposed topology is shown in Figure 4. As observed, the diagram represents the interaction between components of a web application that consumes services from an infrastructure deployed in the AWS cloud.

Using IriusRisk, and based on this topology, automatic threat modeling was also performed through the tool's built-in capabilities. The list of some of the threats generated for the EC2 instances can be seen in Figure 5.

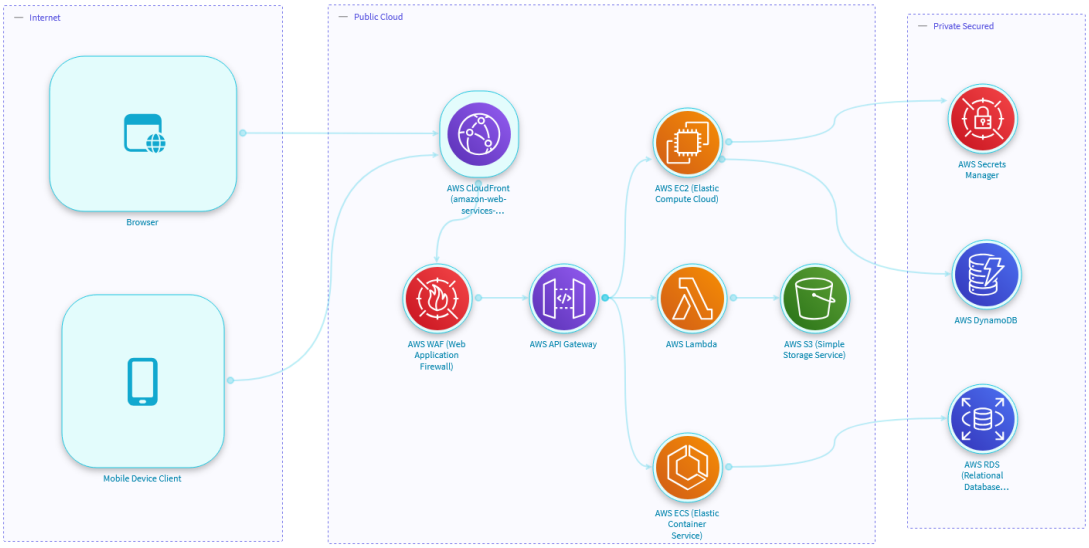

Figure 4: Topology proposed for the system architecture, designed using the IriusRisk tool

0.6.3 Attack Tree Construction

Based on the identification of users and their interactions with the system (subjects and actions) described in the user stories, along with the system topology and the threats associated with components in hypothetical misconfiguration scenarios, the attack tree was manually constructed.

This step was not replicated exactly as in the original paper, since it forms part of the improvements proposed in this thesis work—specifically, the semi-automation of the attack and defense tree generation using Large Language Models (LLMs). As shown in Subsection 0.8.4, attack and defense trees were generated using Large Language Models (LLMs), and an example of a generated attack and defense tree is presented.

0.6.4 Replication of Experiments

Finally, the replication of the results obtained in the practical validation through Security Chaos Engineering (SCE) experiments with ChaosXploit [39], as presented in the ARES paper, was proposed. For this purpose, experiments 3 and 4 were selected, which involved performing privilege escalation and privilege elevation through an EC2 instance and profile.

Initially, the ChaosXploit repository was cloned from GitHub [44], and a professional AWS account was configured by creating the necessary roles and users described in the experiments, as well as the AWS services involved.

Experiment 1: Privilege Escalation

The experiment proposes performing a privilege escalation attack following the technique documented in Hacktricks [45]. An attacker can steal IAM role credentials by querying the metadata endpoint from an EC2 instance that is launched using pre-existing `iam:PassRole` and `ec2:RunInstances` permissions.

Experiment 2: Privilege Elevation

The action carried out in this experiment, privilege elevation, can be seen as a continuation of the privilege escalation experiment. In this case, credential theft is followed by the aggre-

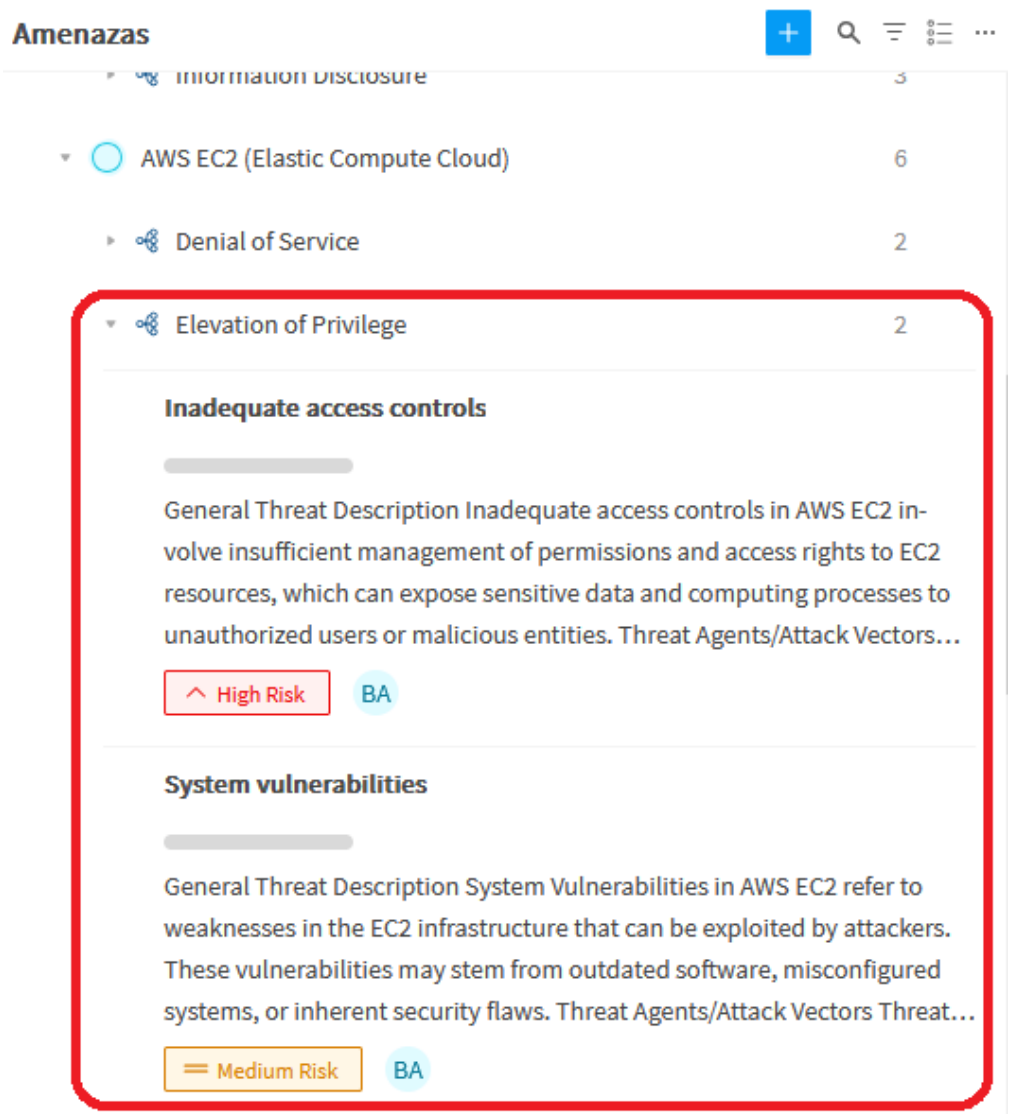

Figure 5: Threat Modeling Report Based on Topology Generated by IriusRisk

gation of additional permissions that the initial cloud insider user did not originally possess. This experiment is documented by Rhino Security Labs [46].

0.7 Replication Experiments of Paper Presented at IJIS

As part of the preliminary exploratory work, experiments 4 and 5 from the paper presented at IJIS were also replicated. The following sections describe the procedures carried out for each experiment.

Experiment 1: Privilege Escalation through AWS CodeBuild

This experiment replicates Experiment 4 (CB) from the IJIS study, where AWS GuardDuty detects privilege escalation attempts originating from AWS CodeBuild. The replication involves reproducing the scenario in which a misconfiguration in the IAM role assigned to a CodeBuild project allows an attacker to obtain elevated privileges across the AWS environment.

Only one of the following permissions is required: `codebuild:StartBuild` or `codebuild:StartBuildBatch`.

With either permission, it becomes possible to define a modified `buildspec` that exfiltrates the raw credentials of the container's IAM role. During replication, events of anomalous service access, credential misuse, and container detention—consistent with those reported in Experiment 4 (CB)—were validated through AWS GuardDuty. The technical reproduction followed the procedure documented in Hacktricks [47].

Experiment 2: SQL Injection in AWS RDS

This experiment replicates Experiment 5 (CP) from the IJIS study, which focuses on detecting vulnerabilities introduced during the CI pipeline using AWS CodeGuru-Security. The replication implements a realistic SQL injection flaw in an application query that interacts with an AWS RDS database.

Consistent with the findings associated with Experiment 5 (CP), AWS CodeGuru-Security successfully identified the vulnerable code pattern and produced the corresponding alerts. The reproduced scenario also aligns with the developer-behavior assumptions from the IJIS work—specifically, unsafely copying generic SQL-handling code without input validation. The resulting impact, including potential database exfiltration, matches the pathways described in the original experiment.

0.8 Contribution to the Research Paper: Anticipating Adversary Behavior in DevSecOps Scenarios through Large Language Models (JNIC 2025)

In line with the proposed specific objectives OBJ2, OBJ3, and OBJ4, I contributed to the development of the research paper titled "Anticipating Adversary Behavior in DevSecOps Scenarios through Large Language Models" [40], which was submitted to JNIC 2025. This work was conducted in collaboration with Mario Marín, a master's student at the University of Murcia in Spain, and the tasks carried out by each contributor are summarized in Table 1. The objectives, key contributions, specific activities, and responsible authors are detailed in the following sections of this document.

Contributor	**Contribution Summary**
Mario Marín	Proposed the workflow diagram for generating attack-defense trees using Large Language Models (LLMs). Defined the evaluation metrics for assessing the quality and structure of the generated attack-defense trees.
Miguel Betancourt	Generated attack-defense trees using various publicly available LLMs. Performed practical validation of the attack path on the EC2 Spot branch of the attack-defense tree through an automated Security Chaos Engineering experiment.

Table 1: Summary of individual contributions to the research paper JNIC 2025

The research conducted for JNIC 2025 focused on the generation of attack trees using Large Language Models (LLMs), aiming to anticipate adversary behavior in DevSecOps scenarios as a tool to support the implementation of a proactive cyber defense strategy.

Our main objective was defined as follows: To enhance the defense and resilience of systems in cyber defense environments by anticipating adversary behavior in DevSecOps scenarios.

The contributions achieved through the development of this work are as follows:

- The design of a new workflow based on Large Language Models (LLMs) for the generation of attack and defense trees.
- A comparative analysis of the performance of different language models in the task of generating attack and defense trees.
- The creation of three metrics that objectively evaluate the performance of language models in this task.
- The practical validation of the generated attack paths through automated Security Chaos Engineering (SCE) experiments.

0.8.1 Proposed Workflow for Attack and Defense Tree Generation Using LLMs

In the pursuit of automating DevSecOps tasks within the software development lifecycle, a workflow based on Large Language Models (LLMs) was proposed for the generation of attack and defense trees. However, the proposed process remains semi-supervised.

Mario Marín, suggested that the generation process using LLMs should begin with initial blocks focused on contextualizing the model. These blocks also define the expected output—in this case, an attack tree in the `.DOT` format.

In the core of the workflow, two loops are executed: the first generates the attack path branch by branch, and the second concatenates the generated branches. In both loops, the flow returns to an evaluation stage based on predefined metrics, which will be discussed in a later section.

Finally, the workflow concludes with blocks dedicated to adjusting the structure and visual formatting of the tree. The complete flow diagram is shown in Figure 6.

0.8.2 Proposed Metrics for Attack and Defense Tree Generation Using LLMs Evaluation

Three evaluation metrics were established to assess the generation of attack and defense trees in a model-agnostic manner, independent of the Large Language Model (LLM) used. These metrics were defined as follows:

- **Mitre_Score**: Indicates the percentage of attack nodes in the tree that are based on documented techniques from the MITRE ATT&CK framework.
- **Ordered_Score**: Evaluates the structure and hierarchy of the generated attack and defense tree.
- **Usable_Score**: Assesses whether each attack node contains the necessary components: a command, an input argument, and an expected result.

The mathematical definitions of these metrics are shown in Table 2.

Metric Name	Formula
Ordered_Score	$Ordered_{Score} = \left(1 - \frac{N_d + N_{sc}}{n}\right) \times 100$
Usable_Score	$Usable_{Score} = \frac{\sum_{i=1}^{n}(C_i + I_i + R_i)}{3n} \times 100$
MITRE_Score	$MITRE_{Score} = \frac{\sum_{i=1}^{n} M_i}{n} \times 100$

Table 2: Evaluation metrics for attack-defense tree generation using LLMs

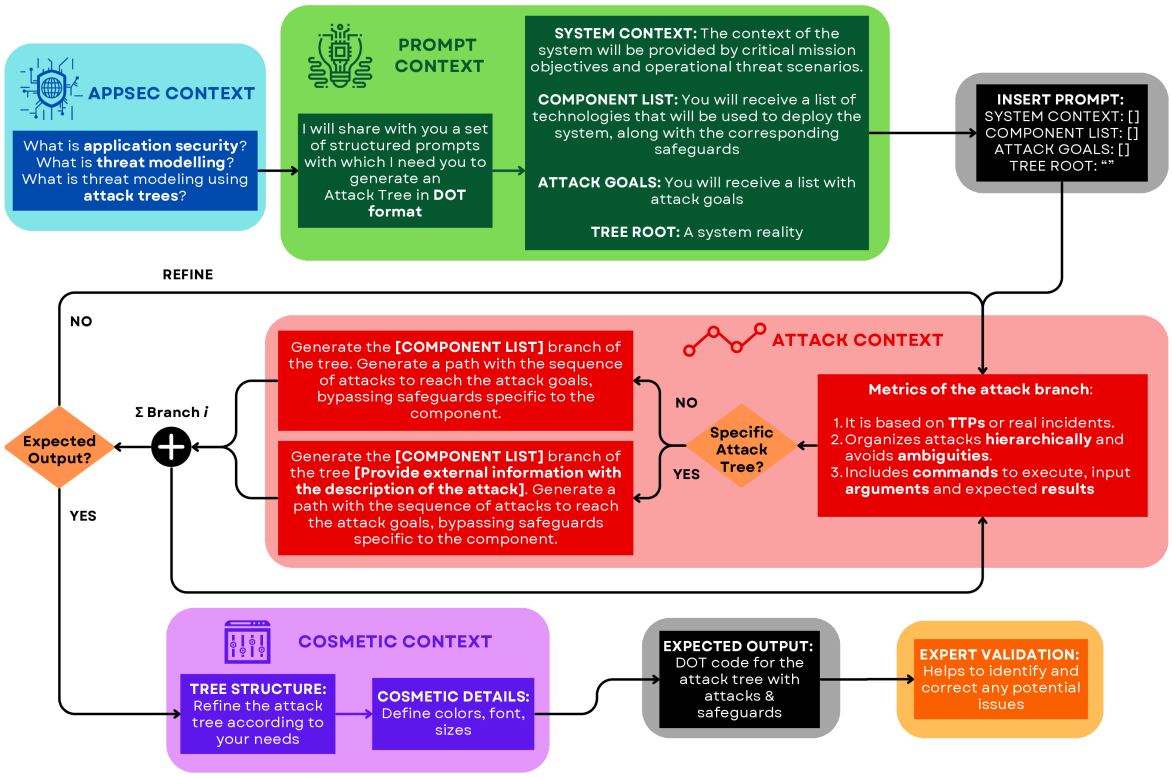

Figure 6: LLM-Based Flow Diagram for Attack and Defense Tree Generation Proposed in paper presented at IJIS 2025

0.8.3 LLMs Used for Attack Tree Generation

The Large Language Models used for the generation of attack trees were ChatGPT [31], Qwen [32], and DeepSeek [33]. Although DeepSeek was not included in the final set of trees submitted with the paper, it was used during the testing phase for tree generation and in refining the prompt set.

In this first phase, the work focused exclusively on generating attack and defense trees using publicly available, general-purpose language models. At this stage of the project, the exploration of cybersecurity-specialized LLMs was planned as future work.

0.8.4 Generated Attack Trees

Figure 8 shows one of the attack and defense trees generated by ChatGPT-4 following the proposed workflow and incorporating the newly defined evaluation metrics. This output allowed us to confirm that it is feasible to automatically generate attack and defense trees using LLMs, while introducing a refinement step based on metric-driven evaluation. As a result, the process produces graphs that are easier to interpret and translate into SCE experiments.

The trees generated by both LLMs were evaluated based on the three key metrics proposed in Table 2. Overall, as we can appreciate in Table 3, a comparison of final scores reveals that QWQ-32B achieved a $Tree_{Score} = 71.60\%$, while GPT-4 scored a $Tree_{Score} = 61.73\%$. In mapping real-world TTPs from the MITRE ATT&CK framework, QwQ-32B in Figure 7 shows a more reliable representation with minimal hallucination, compared to GPT-4, which often provides incomplete or incorrect information. Both models excel in creating hierarchical structures for attack procedures, but QwQ-32B offers clearer paths and priorities. Importantly, QwQ-32B outperforms GPT-4 in providing usable attack procedures, offering more detailed commands and expected outcomes. Having all this information considered,

this makes QwQ-32B the preferred choice for subsequently detailing an SCE experiment from this tree.

Metric	QwQ-32B	GPT-4
Attacks Based on Known TTPs	Reliably maps real-world TTPs from MITRE ATT&CK with minimal hallucination (22.22%) (4/18 nodes).	Most TTPs are incomplete or wrong, slightly reducing realism (11.11%) (1/9 nodes).
Ordered Attack Procedures	Clear hierarchical structure with unambiguous paths and well-defined priority (100%).	Hierarchical but occasionally introduces minor ambiguities in node relationships (100%).
Usable Attack Procedures	Highly detailed commands, inputs, parameters, and expected outcomes (92.59%) (50/18 nodes).	Provides usable steps but lacks technical depth in some areas (74.07%) (20/9 nodes).

Table 3: Comparative table of obtained metrics

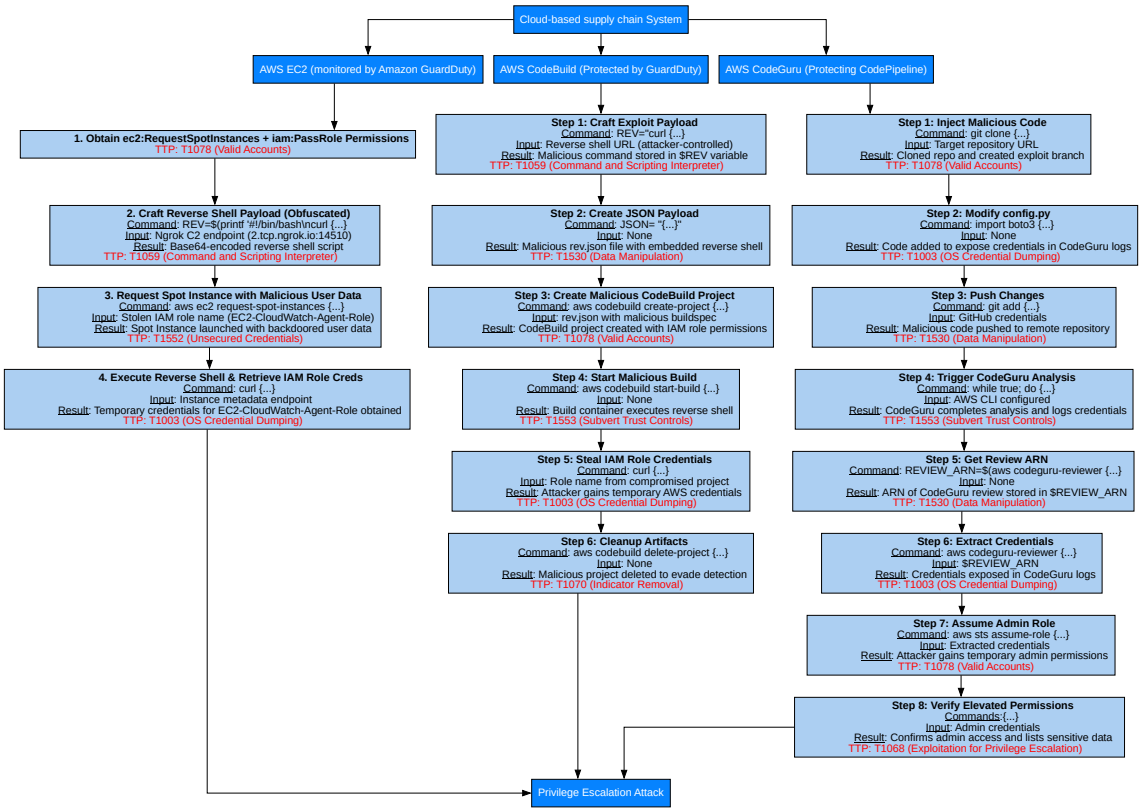

Figure 7: Reduced version of the attack-defense tree generated by QwQ-32B

0.8.5 Proposed Experiments Based on Generated Attack Trees

The practical validation of the attack paths generated by the LLMs was conducted through automated experiments following the Security Chaos Engineering (SCE) methodology. Build-

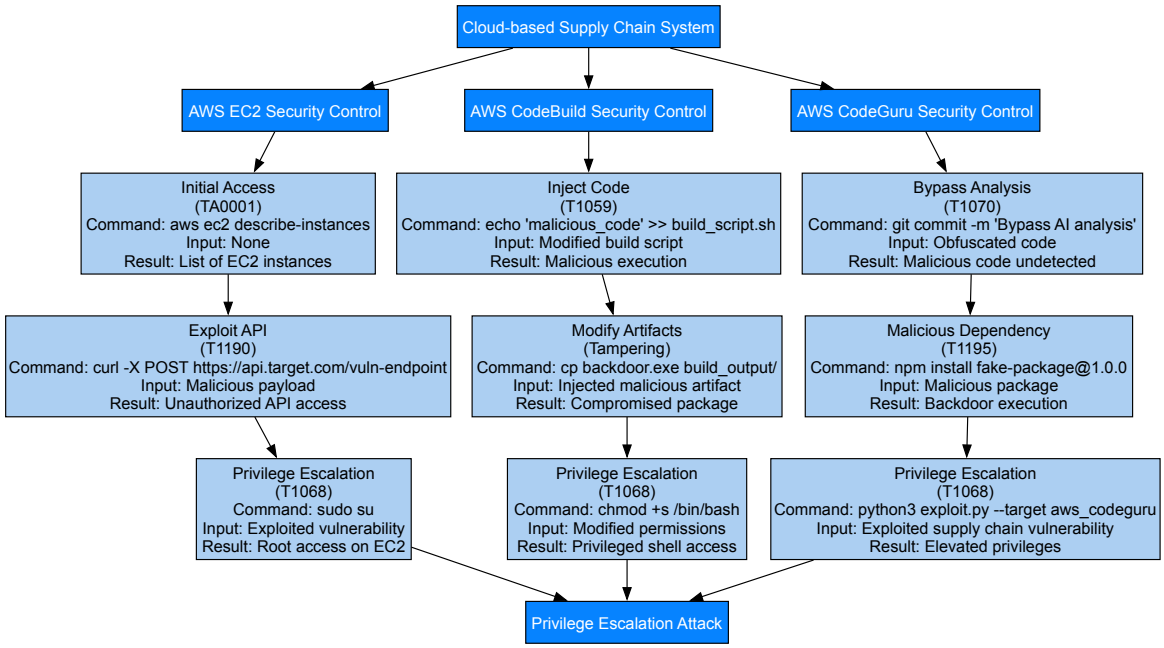

Figure 8: Reduced version of the attack-defense tree generated by ChatGPT-4

ing upon the reference work presented at the ARES Conference and the IJIS paper, I proposed continuing to use *ChaosXploit* to conduct an SCE experiment involving a privilege escalation scenario, specifically targeting the EC2 attack branch of the generated attack tree.

The proposed experiment is based on the first branch of the attack-defense tree generated using ChatGPT-4.

The experiment demonstrates how an insider, leveraging an existing role with the following permissions: `ec2:RequestSpotInstances, iam:PassRole` could launch a Spot EC2 instance. Through the setup of a reverse shell tunnel, the attacker would then gain control of the instance and execute post-exploitation commands such as listing system users.

Initially, the .json file required to automate the experiment was constructed. Figure 11 shows the proposed experiment file, which begins with a description of the attack, followed by an object that defines the steady-state conditions and the corresponding hypothesis.

Within the hypothesis definition, a "probe" test is specified to verify that the system is indeed in a steady state before executing the attack.

In addition to the steady state and hypothesis, the `.json` file also includes a list of *action* methods used to perform the attack. These actions rely on functions implemented using underlying libraries such as `boto3` [48] and `botocore` [49], which enable programmatic interaction with AWS services.

In the automated experiment the 'GuardDuty-Findings' probe step invokes the 'find-changes' function, which checks for updates on GuardDuty alert statuses contained in the 'selected.json' file and returns a boolean value (true if no security alerts are found). The initial condition is validated: if no alerts are detected, the process continues with the next actions.

The next is the action step 'Creating-Instance' that creates the Spot EC2 instance using the "create-instance-spot" function. Such a function takes as relevant arguments the profile information 'EC2-CloudWatch-Agent-Role', the security group ID, and the key pair. The log of the instance creation can be seen in Figure 9. Initially, the Spot EC2 creation command is executed, and the reverse tunnel listening is started on the attacker's machine.

The process waits for confirmation that the victim machine (the newly created Spot EC2 instance) has established communication through the tunnel. Once confirmed the commu-

nication through the tunnel, the third and final step is executed, where the credentials are extracted using the previously mentioned command and utilized to perform tasks that the attacker's initial permissions did not allow. For example, listing users, as shown in Figure 10.

```
[2025-03-17 21:13:35 INFO] Playing your experiment's method now...
[2025-03-17 21:13:35 INFO] Action: Creating_instance
ami-00475f7c6f4a7e5cd
[+] Spot Instance created with ID: i-064e02be18c*****
[+] Waiting 60s, executing the payload...
Executing netcat to listen on port 12345...
Waiting for remote connection in 0.0.0.0:12345...
Established connection with ('127.0.0.1', 42184)
Saving the output in remote_output.json...
Time-Out. Closing connection.
The remote connection its closed.
Netcat process has been stopped.
```

Figure 9: Creating instance EC2 Spot (Log)

```
[2025-03-17 21:14:59 INFO] Action: *******Extracting/Using credentials*******
The file 'remote_output.json' has been successfully cleaned.
TOKEN EX-FILTERED: ASIA5WLTSZADWZGOSL3Y****************...
-----Reading new credentials-----
SUCCESS!
- Arn: arn:aws:iam::941377******:user/usuarioEjemplo1
  CreateDate: 2024-10-30 04:39:17+00:00
  PasswordLastUsed: 2025-03-09 16:05:14+00:00
  Path: /
  UserID: AIDA5WLTSZA********
  UserName: usuarioEjemplo1
- Arn: arn:aws:iam::94137******:user/usuarioEjemplo2
  CreateDate: 2024-12-12 04:27:43+00:00
  PasswordLastUsed: 2025-02-02 08:26:29+00:00
  Path: /
  UserID: AIDA5WLTSZA********
  UserName: usuarioEjemplo2
```

Figure 10: Extracting and using credentials with the instance EC2 Spot created (Log)

0.9 Contribution to the Extended Research Paper: Anticipating Adversary Behavior in DevSecOps Scenarios through Large Language Models (JNIC 2025 Extended)

At the time of presenting this thesis, the extended JNIC paper is still under development. However, work has already been carried out on the following improvement items, which are detailed in this section:

- Creation of new quality validation metrics for attack and defense trees.
- Creation of quality validation metrics for SCE experiments.
- Adjustment of the workflow to enable the automation of attack and defense tree generation and SCE experiment creation.

```
{
    "title": "Privelege Scalation",
    "description": "A set of policies misconfigured that allows an attacker to launch a new
    "steady-state-hypothesis": {
        "title": "AWS GuardFuty detects the use of an instance profile credentials",
        "probes": [
            {
                "type": "probe",
                "name": "GuarDuty-Findings",
                "tolerance": true,
                "provider": {
                    "type": "python",
                    "module": "chaosaws.ec2.probes_ec2_spot",
                    "func": "find_changes",
                    "arguments": {
                        "input_file": "selected.json"
                    }
                }
            }
        ]
    },
    "method": [
        {
            "type": "action",
            "name": "Creating_instance",
            "provider": {
                "type": "python",
                "module": "chaosaws.ec2.actions_spot",
                "func": "create_instance_spot",
                "arguments": {
                    "image_id": "ami-0047[redacted]",
                    "instance_type": "t2.micro",
                    "iam_instance_profile": "EC2-CloudWatch-Agent-Role",
                    "key_name": "parClavesEjemplo1",
                    "security_group_ids": "sg-0ed1[redacted]",
                    "max_count": 1,
                    "min_count": 1
                }
            }
        },
        {
            "type": "action",
            "name": "*******Extracting/Using credentials*******",
            "provider": {
                "type": "python",
                "module": "chaosaws.ec2.actions_spot",
                "func": "extract_credentials_file",
                "arguments": { "keys": "salida_remota.json" }
            }
        }
    ],
    "rollbacks": []
}
```

Figure 11: Experiment Definition `.json` File

0.9.1 New Attack and Defense Tree Quality Metrics

Initially, the metrics used to validate the quality of attack and defense trees consisted of three criteria summarized in Table 2. However, in the first version of the paper JNIC 2025, the generated attack trees showed room for improvement. Although the trees satisfied the minimum requirements regarding hierarchy and node content—such as the number of child nodes associated with a parent node or the definition of the TTP described in each attack node—the overall structure and clarity were still limited.

The new metrics were proposed with the objective of enhancing the readability and reproducibility of both attack and defense branches. These metrics aim to ensure that each node contains accurate, complete, and actionable information, thereby facilitating the replication of attacks and the evaluation of defenses. The new proposed metrics are presented below:

Usable defense procedures

Includes descriptions to execute, input arguments and expected results. Defense nodes must clearly describe what is the classification, action and the test to use:

$$Usable_{Score} = \frac{\sum_{i=1}^{n}(CN_i + A_i + T_i)}{3n} \times 100 \tag{1}$$

- $CN_i = 1$ if the node contains the classification, 0 otherwise.
- $A_i = 1$ if the node contains the action parameters with the technologies related, 0 otherwise.
- $T_i = 1$ if the node contains the test parameters with the technologies related, 0 otherwise.
- $n =$ Total number of attack nodes in the tree

Completeness attack procedures

It follows the five common phases in attack procedures documented in frameworks such as MITRE ATT&CK, the Cyber Kill Chain, and the NIST Cybersecurity Framework. These phases are as follows: Initial Access, Persistence, Execution of Malicious Tasks (including evasion of detection, lateral movement, and command and control), Discovery of Critical Data and Final Attack.

$$Completeness_{Score} = \min(NF, \sum_{i=1}^{N} F_i) * \frac{100}{NF} \tag{2}$$

- F_i Represents the number of attack phases in which a given node i is involved.
- NF Represents the number of attack phases, based on cybersecurity frameworks, considered to describe a branch in the attack-defense tree.
- $N =$ Number of nodes that are not a root or final node.

Documentation and Dependencies

At the top of each branch of the attack-defense tree, there is a node that outlines the prerequisite requirements, such as permissions or configurations.

$$Dependencies_{Score} = \frac{(C_i + P_i + V_i)}{3} \times 100 \tag{3}$$

- $C_i = 1$ if the node contains configurations appropriate to the attack being carried out, 0 otherwise.

- $P_i = 1$ if the node contains the necessary permissions to execute the attack step, 0 otherwise.

- $V_i = 1$ if the node contains descriptive verifications, 0 otherwise.

Scalability and Modularity

The attack procedure described in the attack-defense tree can be applied to different environments or combined with other attack vectors. This makes it reusable in various experiments.

$$SM_{Score} = \frac{\sum_{i=1}^{N}(AB_i + MB_i + (\frac{AN_i}{N}) + (\frac{MN_i}{N}))}{4} \times 100 \tag{4}$$

- $AB_i = 1$ if there is a diversity of branches in an attack-defense tree, it means that there are multiple different compromise techniques that an attacker could exploit to achieve the attack goal. 0 if no diversity is present, only a single technique is available.

- $MB_i = 1$ if it is possible to use the compromise technique in another vector attack, 0 if it is not possible.

- AN_i = Number of nodes that can be adaptable. They could replace their structure to achieve the same function.

- MN_i = Number of nodes that can be used in another compromise technique.

- N = Number of nodes that are not a root or final node.

Operational Security Considerations

Nodes that execute tasks to insert, modify, update, or delete system data or configurations must include appropriate rollback and cleanup commands.

$$Operational_{Score} = \frac{\sum_{i=1}^{N}(R_i)}{N} \times 100 \tag{5}$$

- $R_i = 1$ if the node includes a rollback appropriate, 0 otherwise.

- N = Number of nodes that are not a root or final node, and execute modify, insert, update, or delete operations.

Finally, we can compute the overall score of the tree with the following equation:

$$Tree_{Score} = \frac{\sum_{i=1}^{n} Score_i}{7} \tag{6}$$

- $Score_i$: Represents $MITRE_{Score}$, $Ordered_{Score}$, $Usable_{Score}$, $Completeness_{Score}$, $Dependencies_{Score}$, SM_{Score} and $Operational_{Score}$.

0.9.2 SCE Experiment Quality Metrics

The quality validation metrics proposed for the SCE experiment, as previously discussed, were integrated into the workflow to enable the assessment and refinement of automatically generated SCE experiments. The goal of this integration is to produce experiments that accurately correspond to each branch of the attack and defense tree and that can additionally be implemented as a complementary task within the secure software development lifecycle. Furthermore, by delegating the generation of SCE experiments to the LLM, the entire process becomes automated, reducing manual intervention and increasing the overall efficiency of the workflow.

An SCE experiment consists of a hypothesis, the observable conditions, the steady state, and the specific attack to be executed.

Hypothesis Quality The hypothesis of the SCE experiment must include: (1) a clearly defined independent variable (the attack or vulnerability to be exploited), and (2) a dependent variable (the defensive behavior expected to occur after attack execution).

Attack (AT)	**Alert / Finding (AL)**	**DSR**
Hardcoded Secrets in Code	Hardcoded credential found in source code	CodeGuru
SQL Injection Risk	Unsafe SQL query detected	CodeGuru
Inefficient Resource Use	Inefficient object creation in loop	CodeGuru
Race Condition Risk	Possible race condition on shared variable	CodeGuru
Null Pointer Dereference	Potential null pointer dereference detected	CodeGuru
Poor Exception Handling	Suppressed exception in catch block	CodeGuru
Ineffective Input Validation	Unvalidated user input in handler	CodeGuru
Insecure Hashing Algorithm	Use of MD5 or other insecure hash	CodeGuru
Missing Encryption	Sensitive data stored unencrypted	CodeGuru
Excessive IAM Permissions	Over-permissioned role in SDK call	CodeGuru
Credential Exfiltration	UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration	GuardDuty
Unusual Geolocation Login	UnauthorizedAccess:IAMUser/MaliciousIPCaller	GuardDuty
Privilege Escalation Attempt	UnauthorizedAccess:IAMUser/PrivilegeEscalation	GuardDuty
Suspicious Command Execution	Execution:Runtime/SuspiciousCommand	GuardDuty
Reverse Shell Activity	Execution:Runtime/ReverseShell	GuardDuty
Data Exfiltration	Exfiltration:EC2/UnusualDataTransfer	GuardDuty
Reconnaissance	Recon:EC2/Portscan	GuardDuty
Denial-of-Service Activity	DoS:EC2/UDPReflection	GuardDuty
Ransomware Behavior	Behavior:EC2/EncryptionTool	GuardDuty
Malware Domain Communication	Trojan:EC2/DGADomainRequest	GuardDuty

Table 4: Mapping between attacks, findings, and defensive system resources

$$HypothesisQ_{Score} = \frac{TSR + AT + DSR + AL}{4} \times 100 \tag{7}$$

- $TSR = 1$ if the hypothesis defines the Targeted System Resource; 0 otherwise.
- $AT = 1$ if the hypothesis describes the attack (matching Table 4 or equivalent); 0 otherwise.
- $DSR = 1$ if a defensive system resource is specified; 0 otherwise.
- $AL = 1$ if an expected alert/log is included; 0 otherwise.

Observable Conditions Quality To ensure proper observability, the experiment must define measurable and verifiable alerts or logs expected after attack execution.

$$ConditionsQ_{Score} = \frac{AL}{EAL} \times 100 \tag{8}$$

- AL: Number of alerts/logs listed in the experiment.
- EAL: Number of alerts/logs expected according to Table 4.

Steady State Quality The steady state must explicitly define the system's initial and final observable conditions, enabling verification of deviations introduced by the attack.

$$SteadyStateQ_{Score} = \frac{IC + FC}{2} \times 100 \tag{9}$$

- IC (Initial Conditions): 1 if auditable initial conditions are defined; 0 otherwise.
- FC (Final Conditions): 1 if expected post-attack conditions are defined and verifiable; 0 otherwise.

Note: If both IC and FC equal 0, then $SteadyStateQ_{Score} = 0$.

Initial Conditions Metric Initial conditions define a representative steady state and must reflect auditable system behavior before the attack. Final conditions capture expected behavior after execution (e.g., detection, containment, recovery).

$$SM_{Score} = \frac{DSR + PAL}{2} \times 100 \tag{10}$$

- $DSR = 1$ if a defensive system resource responsible for detection is named; 0 otherwise.
- $PAL = 1$ if pre-existing alerts/logs are defined; 0 otherwise.

0.9.3 New Proposed Workflow for Attack and Defense Tree and SCE Experiment Generation Using LLMs

The new workflow proposed in this work is shown in Figure 12. This workflow automates both the generation of attack and defense trees and the creation of Security Chaos Engineering (SCE) experiments. It represents an evolution of the workflow initially presented at JNIC 2025: while preserving the AppSec context and Prompt context stages previously defined, it extends the Attack context stage by incorporating a new validation loop based on quality metrics, together with a refinement process applied to the SCE experiments generated for each branch of the final attack and defense tree.

In addition to the structural changes introduced in the workflow, this final phase also explored the execution of the process using a different LLM from those employed in earlier stages of the research. The selected model for this phase was Claude Sonnet [42]. This model was chosen due to its strong performance in tasks requiring logical reasoning and code generation, as well as its ability to translate abstract attack and defense descriptions into syntactically correct and executable artifacts. These characteristics proved particularly suitable for the automated generation of SCE experiments, which require both conceptual consistency with the attack and defense tree and precise implementation details compatible with existing chaos engineering frameworks.

Regarding the expected outputs, the system now produces not only the `.dot` file corresponding to the generated attack and defense tree, but also a `.json` file containing the formal definition of the SCE experiment generated through this extended workflow. Figure 13 illustrates a representative branch of the final attack and defense tree, while Figure 14 presents the corresponding SCE experiment generated for that branch.

0.9.4 Final Generated Attack and Defense Tree

The first branch of the final attack and defense tree obtained through the application of the extended quality validation metrics—presented in Table 5—is shown in Figure 13. This branch is composed of multiple attack steps, represented by red-colored nodes, and their corresponding countermeasures or defense steps, represented by blue-colored nodes.

When compared to the attack and defense trees presented in Figures 8 and 7, the improved clarity of the countermeasure nodes associated with each attack step becomes evident. Moreover, the content of each node provides sufficiently detailed and actionable information, enabling a seamless transition toward the subsequent transformation of the attack and defense tree into a Security Chaos Engineering (SCE) experiment.

0.9.5 Final Generated SCE Experiment

Figure 14 shows the final Security Chaos Engineering (SCE) experiment generated by the proposed workflow. With this artifact as the workflow output, it was only necessary to create a virtual environment with the required ChaosToolKit and AWS dependencies installed, and to configure the simulated Secure Software Development Life Cycle (SSDLC) architecture illustrated in Figure 2. Figure 15 presents the execution of the experiment once the environment was properly configured and the necessary users, roles, and policies for the attack scenario simulation were created.

The experiment log shows that the probes defined in the generated `.json` file are executed first to validate the stated hypothesis. Once the expected result is obtained—specifically, an `AccessDenied` response when attempting to perform a `PassRole` operation in the presented log—the workflow proceeds with the execution of the defined actions. These actions provide practical validation that the simulated military supply chain system is correctly configured to withstand the attack behavior described in the corresponding attack and defense tree.

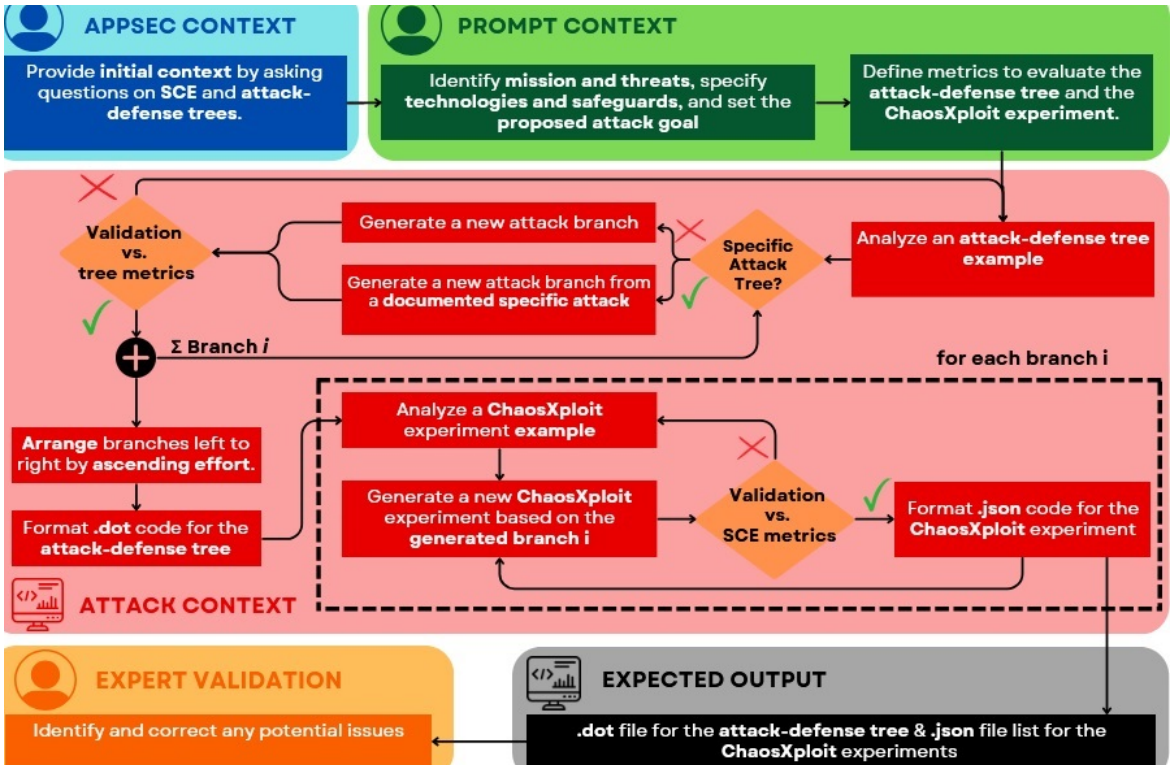

Figure 12: New Proposed Workflow for Attack and Defense Tree and SCE Exper-iment Generation Using LLMs

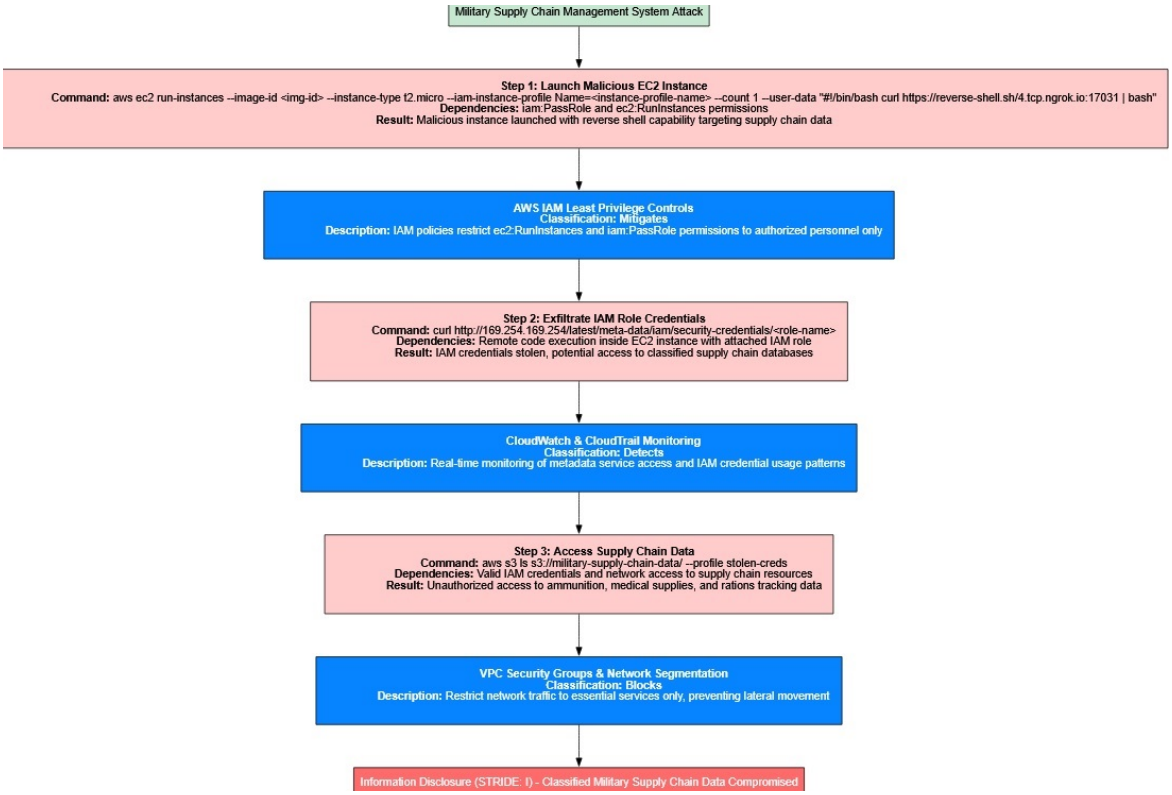

Figure 13: First Branch of the final attack and defense tree generated

0.10 Results Analysis

The main outcome of this work is a workflow for the semi-automatic generation of attack and defense trees, as well as for the construction of Security Chaos Engineering (SCE) experiments (Figure 12), corresponding to the improvements achieved in Phase 3 of the proposed methodology. In Subsection 0.4.2, associated with Phase 1, the work initially focused on replicating prior approaches that addressed isolated and sequential tasks for attack and defense tree generation, such as entity recognition in user stories using *Named Entity Recognition* (NER) models or the use of tools like IriusRisk to extract attack vectors from manually modeled topologies. However, these approaches were characterized by disconnected processes and a high degree of human intervention in both threat modeling and the definition and execution of SCE experiments.

Phase 1 concluded with the design of an integrated workflow for attack and defense tree generation, along with the definition of specific metrics for quality validation. This represented a significant improvement over previous works, as it introduced an objective mechanism for assessing the quality of the generated trees and enabled a comparative evaluation of different LLMs in the tree generation task, as shown in Table 3. The quality evaluations of attack and defense trees were conducted in Phase 2, during which a noticeable reduction in model hallucination effects was observed. Although only the final attack and defense trees are presented in this document (Figures 8 and 7), multiple intermediate graphs were generated during experimentation that did not satisfy the formal structure of a tree or the expected semantic and logical coherence of attack vectors.

Additionally, Phase 3 introduced substantial enhancements to the initial workflow through the extension and refinement of the quality metrics applied to attack and defense trees. These extended metrics enabled the evaluation of not only structural and basic coherence properties, but also aspects related to interpretability and reproducibility, ensuring that each node

```
{
    "title": "IAM Least Privilege Enforcement Validation",
    "description": "Validate that IAM policies correctly prevent unauthorized EC2 instance launches with privileged roles in the military supply chain environment",
    "steady-state-hypothesis": {
        "title": "IAM policies should block unauthorized users from launching EC2 instances with supply chain access roles",
        "probes": [
            {
                "type": "probe",
                "name": "verify IAM policy denies ec2:RunInstances for unauthorized users",
                "tolerance": true,
                "provider": {
                    "type": "python",
                    "module": "chaosaws.iam.probes",
                    "func": "check_policy_allows_action",
                    "arguments": {
                        "user_name": "test-unauthorized-user",
                        "action": "ec2:RunInstances",
                        "resource": "arn:aws:ec2:*:*:instance/*"
                    }
                }
            },
            {
                "type": "probe",
                "name": "verify IAM role cannot be passed by unauthorized users",
                "tolerance": false,
                "provider": {
                    "type": "python",
                    "module": "chaosaws.iam.probes",
                    "func": "check_policy_allows_action",
                    "arguments": {
                        "user_name": "test-unauthorized-user",
                        "action": "iam:PassRole",
                        "resource": "arn:aws:iam::*:role/SupplyChainAccessRole"
                    }
                }
            },
            {
                "type": "probe",
                "name": "Check CloudTrail logging is active for IAM events",
                "tolerance": true,
                "provider": {
                    "type": "python",
                    "module": "chaosaws.cloudtrail.probes",
                    "func": "is_trail_logging",
                    "arguments": {
                        "trail_name": "supply-chain-audit-trail"
                    }
                }
            }
        ]
    },
    "method": [
        {
            "type": "action",
            "name": "Attempt unauthorized EC2 instance launch with privileged role",
            "provider": {
                "type": "python",
                "module": "chaosaws.ec2.actions",
                "func": "run_instance",
                "arguments": {
                    "image_id": "ami-0abcdef1234567890",
                    "instance_type": "t2.micro",
                    "iam_instance_profile": "SupplyChainAccessProfile",
                    "user_data": "#!/bin/bash\necho 'Chaos Engineering Test Instance'",
                    "aws_profile": "chaos-test-user"
                }
            }
        },
        {
            "type": "action",
            "name": "Verify access denied event is logged",
            "provider": {
                "type": "python",
                "module": "chaosaws.cloudtrail.actions",
                "func": "check_recent_events",
                "arguments": {
                    "event_name": "RunInstances",
                    "error_code": "UnauthorizedOperation",
                    "time_range": "5"
                }
            }
        }
    ],
    "rollbacks": [
        {
            "type": "action",
            "name": "terminate any created test instances",
            "provider": {
                "type": "python",
                "module": "chaosaws.ec2.actions",
                "func": "terminate_instances",
                "arguments": {
                    "filters": [
                        {
                            "Name": "tag:Purpose",
                            "Values": ["ChaosEngineering"]
                        }
                    ]
                }
            }
        },
        {
            "type": "action",
            "name": "Clean up test IAM resources",
            "provider": {
                "type": "python",
                "module": "chaosaws.iam.actions",
                "func": "delete_user",
                "arguments": {
                    "user_name": "test-unauthorized-user"
                }
            }
        }
    ]
```


Figure 14: SCE Experiment of First Branch Attack and Defense Tree Generated With Workflow Extended

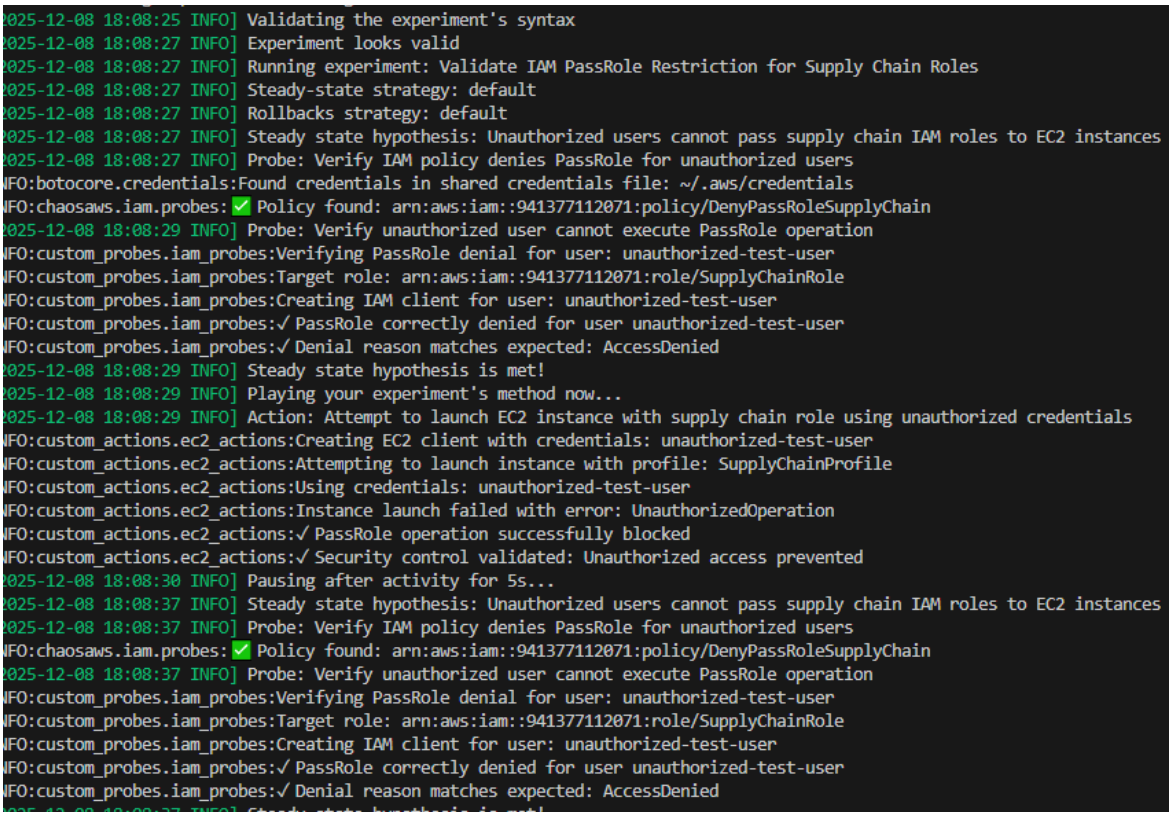

Figure 15: SCE Experiment detailed log

contained sufficient and actionable information to understand and execute the corresponding attack or defense step. Furthermore, Phase 3 incorporated, for the first time, dedicated quality validation metrics for SCE experiments generated from the trees, as shown in Table 5. This addition directly improved the consistency, executability, and reliability of the produced experiments. Taken together, these extensions strengthen both the quality of threat modeling and the robustness of security experiment execution in controlled environments.

Finally, the results obtained in this work enable the generation of attack and defense trees such as the one shown in Figure 13, and the automatic derivation of SCE experiments—defined through `.json` files—such as the example illustrated in Figure 14. These experiments can then be executed in a controlled environment, as shown in Figure 15.

Metric	Equation	Description
MITRE_Score	$MITRE_{Score} = \frac{\sum_{i=1}^{n} M_i}{n} \times 100$	Evaluates whether the nodes correspond to real techniques based on the MITRE ATT&CK framework.
Ordered_Score	$Ordered_{Score} = \left(1 - \frac{N_d + N_{sc}}{n}\right) \times 100$	Evaluates whether the attack procedures are correctly ordered and hierarchically structured.
Usable_Score	$Usable_{Score} = \frac{\sum_{i=1}^{n}(C_i + I_i + R_i)}{3n} \times 100$	Measures whether the nodes contain executable commands, required input arguments, and expected results.
Completeness_Score	$Completeness_{Score} = \min(NF, \sum_{i=1}^{N} F_i) \times \frac{100}{NF}$	Evaluates whether the key attack phases are covered according to frameworks such as ATT&CK, the Cyber Kill Chain, or the NIST CSF.
Dependencies_Score	$Dependencies_{Score} = \frac{C_i + P_i + V_i}{3} \times 100$	Evaluates whether the root node of a branch contains the required configurations, permissions, and verifications.
SM_Score	$SM_{Score} = \frac{\sum_{i=1}^{N}\left(AB_i + MB_i + \frac{AN_i}{N} + \frac{MN_i}{N}\right)}{4} \times 100$	Measures modularity and scalability: node reuse, diversity, and adaptability of the tree.
Operational_Score	$Operational_{Score} = \frac{\sum_{i=1}^{N} R_i}{N} \times 100$	Evaluates whether the nodes include appropriate rollback or cleanup procedures.
Tree_Score	$Tree_{Score} = \frac{\sum_{i=1}^{7} Score_i}{7}$	Overall average of all attack-defense tree metrics.

Table 5: Attack-Defense Trees Extended Metrics

0.11 Conclusions and Future Works

The work developed over these two semesters enabled the validation, extension, and automation of previous proposals at the intersection of DevSecOps, SCE, and artificial intelligence, demonstrating that LLMs can effectively support the anticipation of adversary behavior, the creation of experimentation scenarios, and the continuous improvement of the Secure Software Development Lifecycle (SSDLC).

- Proposed a language-model-based (LLM) workflow for generating attack–defense trees representing adversary behavior.
- Defined key metrics to evaluate the quality of the generated attack–defense trees, reducing model hallucination and ensuring the required hierarchical structure.
- Constructed an SCE experiment using one of the selected branches of the attack–defense tree.
- Executed the SCE experiment in ChaosXploit, exploiting different attack vectors and implementing the corresponding defense mechanisms within a cloud-based test environment.
- Demonstrated that the generated attack–defense trees allow anticipation of adversary behavior by exploiting pre-existing vulnerabilities and support effective cyber-defense strategies.
- Contributed to the paper presented at JNIC 2025 [40, 41] by generating attack–defense trees, defining evaluation metrics, and validating attack paths derived from LLMs.
- Contributed to the extended version of the research (currently under development), including: i) Definition of new evaluation metrics for LLM-generated attack–defense trees and SCE experiments, ii) Formulation of a modified workflow enabling a fully automated end-to-end process.

As future work, we propose:

- Incorporating pre- and post-execution quality metrics for SCE experiments generated from LLM-derived attack trees.
- Modifying the branch-generation process so that branches originate from defense nodes (classified as detective, corrective, or preventive) rather than attack nodes.
- Creating attack vectors designed to challenge existing countermeasures, shifting the methodology from hypothetical attack paths to realistic adversarial scenarios grounded in deployed defenses, enabling a proactive security validation strategy.

Bibliography

[1] Maurice Dawson, Darrell Norman Burrell, Emad Rahim, and Stephen Brewster. Integrating software assurance into the software development life cycle (sdlc). *International Journal of Secure Software Engineering*, 1(3):1–15, 2010. Accessed: 2025-11-24.

[2] Isaac Chin Eian, Lim Ka Yong, Majesty Yeap Xiao Li, Noor Affan Bin Noor Hasmaddi, and Fatima tuz Zahra. Integration of security modules in software development lifecycle phases. 2020.

[3] EY Financial Services Thought Gallery. Giss cyber-security report update. `https://eyfinancialservicesthoughtgallery.ie/wp-content/uploads/2018/11/GISS__Cyber-security-report-update_Digital.pdf`, 2018. Accessed: 2025-11-24.

[4] A. Chidukwani, S. Zander, and P. Koutsakis. A survey on the cyber security of small-to-medium businesses: Challenges, research focus and recommendations. *IEEE Access*, 10:85701–85719, 2022.

[5] Peter Mell and Timothy Grance. The nist definition of cloud computing. Technical Report SP 800-145, National Institute of Standards and Technology, 2011.

[6] U.S. Securities and Exchange Commission. Order instituting cease-and-desist proceedings pursuant to section 8a of the securities act of 1933 and section 21c of the securities exchange act of 1934, making findings, and imposing a cease-and-desist order. Technical Report Release No. 10485; Release No. 83096; File No. 3-18448, U.S. Securities and Exchange Commission, April 2018. In the Matter of Altaba Inc., f/d/b/a Yahoo! Inc.

[7] Ugur Koc, Parsa Saadatpanah, Jeffrey S. Foster, and Adam A. Porter. Learning a classifier for false positive error reports emitted by static code analysis tools. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, MAPL 2017, page 35–42, New York, NY, USA, 2017. Association for Computing Machinery.

[8] Zoltán Szabó and Vilmos Bilicki. A New Approach to Web Application Security: Utilizing GPT Language Models for Source Code Inspection. *Future Internet*, 15(10), 2023.

[9] Maanak Gupta, Charankumar Akiri, Kshitiz Aryal, Eli Parker, and Lopamudra Praharaj. From ChatGPT to ThreatGPT: Impact of Generative AI in Cybersecurity and Privacy. *IEEE Access*, 11:80218–80245, 2023.

[10] Timothy McIntosh, Tong Liu, Teo Susnjak, Hooman Alavizadeh, Alex Ng, Raza Nowrozy, and Paul Watters. Harnessing GPT-4 for generation of cybersecurity GRC policies: A focus on ransomware attack mitigation. *Computers & Security*, 134:103424, 2023.

[11] Madhav Nair, Rajat Sadhukhan, and Debdeep Mukhopadhyay. How Hardened is Your Hardware? Guiding ChatGPT to Generate Secure Hardware Resistant to CWEs. In Shlomi Dolev, Ehud Gudes, and Pascal Paillier, editors, *Cyber Security, Cryptology, and Machine Learning*, pages 320–336, Cham, 2023. Springer Nature Switzerland.

[12] Matija Cankar, Nenad Petrovic, Joao Pita Costa, Ales Cernivec, Jan Antic, Tomaz Martincic, and Dejan Stepec. Security in DevSecOps: Applying Tools and Machine Learning to Verification and Monitoring Steps. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*, ICPE '23 Companion, page 201–205, New York, NY, USA, 2023. Association for Computing Machinery.

[13] Olga Gadyatskaya and Dalia Papuc. ChatGPT Knows Your Attacks: Synthesizing Attack Trees Using LLMs. In Chutiporn Anutariya and Marcello M. Bonsangue, editors, *Data Science and Artificial Intelligence*, pages 245–260, Singapore, 2023. Springer Nature Singapore.

[14] Yadong Zhang, Shaoguang Mao, Tao Ge, Xun Wang, Adrian de Wynter, Yan Xia, Wenshan Wu, Ting Song, Man Lan, and Furu Wei. Llm as a mastermind: A survey of strategic reasoning with large language models. *arXiv preprint arXiv:2404.01230*, 2024.

[15] Alaeddine Diaf, Abdelaziz Amara Korba, Nour Elislem Karabadji, and Yacine Ghamri-Doudane. Beyond detection: Leveraging large language models for cyber attack prediction in iot networks. In *2024 20th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*, pages 117–123. IEEE, 2024.

[16] Ying Zhang, Xiaoyan Zhou, Hui Wen, Wenjia Niu, Jiqiang Liu, Haining Wang, and Qiang Li. Tactics, techniques, and procedures (ttps) in interpreted malware: A zero-shot generation with large language models. *arXiv preprint arXiv:2407.08532*, 2024.

[17] Lingzhi Wang, Zhenyuan Li, Zonghan Guo, Yi Jiang, Kyle Jung, Kedar Thiagarajan, Jiahui Wang, Zhengkai Wang, Emily Wei, Xiangmin Shen, and Yan Chen. From sands to mansions: Simulating full attack chain with llm-organized knowledge, 2024.

[18] Christoforus Yoga Haryanto, Anne Maria Elvira, Trung Duc Nguyen, Minh Hieu Vu, Yoshiano Hartanto, Emily Lomempow, and Arathi Arakala. Contextualized ai for cyber defense: An automated survey using llms. In *2024 17th International Conference on Security of Information and Networks (SIN)*, pages 1–8, 2024.

[19] Johannes F. Loevenich, Erik Adler, Adrien Bécue, Alexander Velazquez, Konrad Wrona, Vasil Boshnakov, Jerry Falkcrona, Nils Nordbotten, Olwen L. Worthington, Juha Röning, Roberto Rigolin, and F. Lopes. Training autonomous cyber defense agents: Challenges & opportunities in military networks. In *MILCOM 2024 - 2024 IEEE Military Communications Conference (MILCOM)*, pages 158–163, 2024.

[20] Optum. Chaoslingr: Introducing security into chaos testing. `https://github.com/Optum/ChaoSlingr`, April 2019. Last time accessed: 2025-12-01.

[21] Kennedy A. Torkura, Muhammad I.H. Sukmana, Feng Cheng, and Christoph Meinel. CloudStrike: Chaos Engineering for Security and Resiliency in Cloud Infrastructure. *IEEE Access*, 8:123044–123060, 2020.

[22] Charalambos Konstantinou, George Stergiopoulos, Masood Parvania, and Paulo Esteves-Verissimo. Chaos Engineering for Enhanced Resilience of Cyber-Physical Systems. In *2021 Resilience Week (RWS)*, pages 1–10, 2021.

[23] Salah Sharieh and Alexander Ferworn. Securing apis and chaos engineering. In *2021 IEEE Conference on Communications and Network Security (CNS)*, pages 290–294, 2021.

[24] Thomas Bailey, Patrick Marchione, Peter Swartz, Raed Salih, Michael Clark, and Robert Denz. Measuring resiliency of system of systems using chaos engineering experiments. In *2022 SPIE 12117, Disruptive Technologies in Information Sciences VI*, volume 1211704, page 26, 2022.

[25] Rodi Jolak, Mazen Mohamad, Ramana Reddy Avula, Jason Meek, and Alexander Å ström. Scene: Guidelines for security chaos engineering based on a systematic literature review. Working paper, SSRN, 2025.

[26] Dhanasekar Elumalai. The role of chaos engineering in devsecops: Strengthening security and compliance in agile. *The American Journal of Engineering and Technology*, 7(06):240–247, 2025. Open Access.

[27] Sara Palacios Chavarro, Pantaleone Nespoli, Daniel Díaz-López, and Yury Niño Roa. On the Way to Automatic Exploitation of Vulnerabilities and Validation of Systems Security through Security Chaos Engineering. *Big Data and Cognitive Computing*, 7(1), 2023.

[28] Martin Bedoya, Sara Palacios, Daniel Diaz-López, Pantaleone Nespoli, Estefania Laverde, and Sebastián Suárez. Securing Cloud-Based Military Systems with Security Chaos Engineering and Artificial Intelligence. In *Proceedings of the 18th International Conference on Availability, Reliability and Security*, ARES '23, New York, NY, USA, 2023. Association for Computing Machinery.

[29] Martin Bedoya, Sara Palacios, Daniel Díaz-López, Estefania Laverde, and Pantaleone Nespoli. Enhancing devsecops practice with large language models and security chaos engineering. *International Journal of Information Security*, 23(6):3765–3788, 2024.

[30] Guntur Budi Herwanto. ner-privacy-engineering. `https://github.com/gunturbudi/ner-privacy-engineering`, 2021. Accessed: 2025-05-31.

[31] OpenAI. ChatGPT: Gpt-4 model, 2023. Accessed via chat.openai.com.

[32] Alibaba Cloud. Qwen: Large language models by alibaba cloud, 2023. Accessed via https://huggingface.co/Qwen on May 30, 2025.

[33] DeepSeek-AI. Deepseek: Open-source language models for reasoning and coding, 2023. Accessed via https://huggingface.co/deepseek-ai on May 30, 2025.

[34] Alyzia-Maria Konsta, Alberto Lluch Lafuente, Beatrice Spiga, and Nicola Dragoni. Survey: Automatic generation of attack trees and attack graphs. *Computers & Security*, 137:103602, 2024.

[35] Attack trees. In Stan Z. Li and Anil K. Jain, editors, *Encyclopedia of Biometrics*. Springer, Boston, MA, 2009.

[36] MITRE Corporation. MITRE ATT&CK Framework. `https://attack.mitre.org/`, 2024. Accessed: 2026-02-15.

[37] Chaos Toolkit. Chaos toolkit — the chaos engineering toolkit for developers. `https://chaostoolkit.org/`, n.d. Consultado: 23 de noviembre de 2025.

[38] Chaos Toolkit Incubator. chaostoolkit-aws / chaosaws. https://github.com/chaostoolkit-incubator/chaostoolkit-aws/tree/master/chaosaws, n.d. Repositorio en GitHub, consultado: 23 de noviembre de 2025.

[39] S. Palacios, D. Díaz-López, and P. Nespoli. Chaosxploit: A security chaos engineering framework based on attack trees. In *VII Jornadas Nacionales de Investigación en Ciberseguridad (JNIC)*, volume 1, pages 130–137, Bilbao, Spain, 2022.

[40] Mario Marín-Caballero, Miguel Betancourt Alonso, Daniel Díaz-López, Ángel Luis Perales Gómez, Pantaleone Nespoli, and Gregorio Martínez Pérez. Anticipating adversary behavior in devsecops scenarios through large language models. In *VIII Jornadas Nacionales de Investigación en Ciberseguridad (JNIC)*, España, 2025. Accedido: junio 2025.

[41] Mario Marín Caballero, Miguel Betancourt Alonso, Daniel Díaz-López, Angel Luis Perales Gómez, Pantaleone Nespoli, and Gregorio Martínez Pérez. Anticipating adversary behavior in devsecops scenarios through large language models. *arXiv preprint arXiv:2602.14106*, 2026.

[42] Anthropic. Claude ai. https://claude.ai, 2025. Accessed: 2025-02-05.

[43] IriusRisk. Iriusrisk - threat modeling tool. https://www.iriusrisk.com/, 2025. Accessed: 2025-05-31.

[44] Sara Palacios. Chaosxploit: A security chaos engineering framework based on attack trees. https://github.com/SaraPalaciosCh/ChaosXploit, 2022. Accedido: junio 2025.

[45] HackTricks. Aws ec2 privilege escalation - iam passrole + ec2 runinstances. https://cloud.hacktricks.wiki/en/pentesting-cloud/aws-security/aws-privilege-escalation/aws-ec2-privesc.html#iam-passrole-ec2-runinstances, 2024. Accedido: mayo 2025.

[46] Nick Spagnola. Weaponizing aws ecs task definitions to steal credentials from running containers. https://rhinosecuritylabs.com/aws/weaponizing-ecs-task-definitions-steal-credentials-running-containers/, 2020. Accedido: junio 2025.

[47] HackTricks. Aws codebuild privilege escalation. https://cloud.hacktricks.wiki/en/pentesting-cloud/aws-security/aws-privilege-escalation/aws-codebuild-privesc.html, 2025. Accedido: junio 2025.

[48] Amazon Web Services. Boto3: The aws sdk for python. https://github.com/boto/boto3, 2024. Accessed: 2025-02-24.

[49] Amazon Web Services. Botocore. https://pypi.org/project/botocore/, 2024. Accessed: 2025-02-24.

0.12 Annexes

0.12.1 Participation in the JNIC 2025

The *Jornadas Nacionales de Investigación en Ciberseguridad* (JNIC) are national academic conferences held annually in different cities across Spain, dedicated to the presentation of research in cybersecurity. To participate in the conference, a research paper must be submitted; in our case, the paper was selected for presentation in the city of Zaragoza on June 4, 2025.

The final version of the paper submitted is included as part of this thesis, representing the main research outcome developed during the first semester of this graduation project.

Anticipating Adversary Behavior in DevSecOps Scenarios through Large Language Models

Mario Marín-Caballero[1], Miguel Betancourt Alonso[2], Daniel Díaz-López[1,2]
Angel Luis Perales Gómez[3] Pantaleone Nespoli[1] Gregorio Martínez Pérez[1]
[1]Department of Information and Communications Engineering, University of Murcia, 30100, Murcia, Spain
{mario.m.c, danielorlando.diaz, pantaleone.nespoli, gregorio}@um.es
[2]School of Engineering, Science and Technology, Universidad del Rosario, Bogotá, Colombia
{miguels.betancourt, danielo.diaz}@urosario.edu.co
[3]Department of Computers Engineering and Technology, University of Murcia, 30100, Murcia, Spain
angelluis.perales@um.es

***Abstract*—The most valuable asset of any cloud-based organization is data, which is increasingly exposed to sophisticated cyberattacks. Until recently, the implementation of security measures in DevOps environments was often considered optional by many government entities and critical national services operating in the cloud. This includes systems managing sensitive information, such as electoral processes or military operations, which have historically been valuable targets for cybercriminals. Resistance to security implementation is often driven by concerns over losing agility in software development, increasing the risk of accumulated vulnerabilities. Nowadays, patching software is no longer enough; adopting a proactive cyber defense strategy, supported by Artificial Intelligence (AI), is crucial to anticipating and mitigating threats. Thus, this work proposes integrating the Security Chaos Engineering (SCE) methodology with a new LLM-based flow to automate the creation of attack-defense trees that represent adversary behavior and facilitate the construction of SCE experiments based on these graphical models, enabling teams to stay one step ahead of attackers and implement previously unconsidered defenses. Further detailed information about the experiment performed, along with the steps to replicate it, can be found in the following repository: https://github.com/mariomc14/devsecops-adversary-llm.git.**

***Index Terms*—Adversary behaviour, LLMs, attack-defence trees, DevSecOps, Security Chaos Engineering (SCE), threat intelligence, Cyber Situational Awareness (CSA), cyber defence**

Contribution type: *Original research*

I. Introduction

Cyber defense refers to the set of security measures and strategies against cyberattacks [1] and can be implemented either actively or passively. Historically, private and public corporations have mostly relied on passive measures, focusing on patching vulnerabilities. However, active cyber defense allows for a broader perspective and the consideration of a wide range of interaction scenarios between individuals, which has proven to be a more effective strategy [2]. This shift from reactive to proactive approaches has led organizations to seek methodologies that can systematically strengthen their security posture.

One such methodology that supports the implementation of a proactive cyber defense strategy is Security Chaos Engineering (SCE), which improves security and tests the resilience of cloud-deployed architectures through fault injection and controlled experimentation [3]. SCE represents a paradigm shift in how organizations approach security, moving from merely responding to threats toward actively testing defenses before attackers can exploit them. This proactive approach aligns perfectly with modern software development practices that emphasize agility and continuous improvement.

DevOps, which refers to the agile and collaborative integration of development and operations teams [4], embodies these modern practices. When security tasks are integrated from the early stages of development through to software delivery, the practice evolves into DevSecOps. This integration creates a comprehensive environment where security becomes an inherent part of the development lifecycle rather than an afterthought or separate concern, thereby strengthening an organization's overall defensive posture.

An optimal cyber defense strategy in a DevSecOps scenario should integrate vulnerability identification, prioritize threats based on their likelihood of occurrence, explore various attack scenarios while anticipating adversary behavior, and implement automated and effective countermeasures. This holistic approach requires sophisticated tools and methodologies that can model complex attacker-defender dynamics within fast-paced development environments.

Attack-defense trees represent one traditional approach to addressing this need, offering a comprehensive visual framework for cybersecurity analysis by combining offensive and defensive actions in a hierarchical structure. By extending traditional attack trees to include defensive countermeasures, these models enable a dynamic representation of the attacker-defender interplay. This structured approach facilitates improved risk assessment, interdisciplinary collaboration, and strategic decision-making in addressing complex, multi-stage security threats within DevSecOps pipelines.

However, when using these methods (considered traditional approaches for anticipating adversary behavior in cyberspace), their effectiveness is limited by the creativity and knowledge of the security teams implementing them. This limitation creates a significant challenge in maintaining both security and agility

in DevSecOps environments, as human-centered analysis can become a bottleneck in rapid development cycles while still potentially missing novel attack vectors.

In response to these limitations, the state-of-the-art in Large Language Models (LLMs) has emerged as a promising solution by enabling the automation of tasks requiring human language comprehension and generation. Some LLMs, such as RepairLlama [5], which corrects code errors, and Hackmentor [6], which applies fine-tuning to pre-existing LLM models, have been trained directly on cybersecurity data, making them specialized models in this field. Others, like ChatGPT [7], are pre-trained on general data and later adapted using fine-tuning techniques or methods such as in-context learning and chain-of-thought reasoning to tackle complex cybersecurity tasks [8].

Unlike RepairLlama and Hackmentor, the great potential of ChatGPT lies not only in its ability to correct code errors, but also in its outstanding performance across various cybersecurity tasks, such as autonomous vulnerability exploration and the discovery of new threats [9], without requiring a prior transfer learning process, making it readily available for immediate use.

The strategic implementation of LLMs to forecast adversarial behavior patterns (specifically prospective attack procedures, their sequential execution patterns (e.g., branch sequencing within attack-defense tree frameworks), and associated cost-benefit considerations) holds significant potential to mitigate cognitive biases inherent in conventional security testing methodologies within DevSecOps pipelines. Furthermore, such an approach facilitates the systematic exploration of novel attack vectors while refining vulnerability detection and mitigation strategies. Operational integration of such predictive systems within cyber command frameworks could enhance proactive threat modeling, enabling stakeholders to preemptively identify and neutralize emerging threats targeting cloud-based national critical infrastructure assets.

Thus, the main contributions of this paper are summarized as follows:

- A new LLM-based flow for generating attack-defense trees that represent adversary behavior.
- A comparative analysis of attack-defense tree generation is conducted using different general-purpose LLMs, providing the models with a set of prompts within a DevSecOps and cyber defense context to evaluate their ability to generate realistic attack structures with a clear hierarchy aligned with the proposed attack objective.
- The definition of key metrics for assessing the quality of attack-defense trees generated by different models. These metrics ensure that the trees are based on real-world attacks and countermeasures, maintain a clear hierarchical structure, and align with a well-defined attack objective.
- The validation of the proposal through the evaluation of some generated attack-defense trees, and the implementation of a SCE experiment that allow to test the feasibility of the generated paths from the perspective of a realistic adversary.

This paper is structured as follows: Section II explores the LLMs used for cybersecurity and cyber defense purposes. Then, Section III describes our proposed method for generating and evaluating attack-defense trees using general-purpose LLM models. Next, in Section IV, the best-obtained attack-defense trees are used to construct and execute our experiments. Finally, Section V presents the conclusions and analyses potential future research directions to improve our proposal.

II. State of the art

This section describes recent advancements in the use of LLMs for cybersecurity and new frontiers in proactive security analysis. In this sense, the work proposed by Yadong Zhang *et al.* [10] provides a comprehensive survey on leveraging LLMs for strategic reasoning—a complex cognitive ability involving decision-making in multi-agent, dynamic, and uncertain environments. The study systematically explores LLMs' applications, methodologies, and evaluation in strategic scenarios, emphasizing their growing significance in multi-agent interactions and decision-making.

Moreover, Alaeddine Diaf *et al.* [11] present a novel framework for IoT networks that leverages LLMs and Long Short-Term Memory (LSTM) models to predict intrusions by analyzing network packets. The framework uses a fine-tuned Generative Pre-trained Transformer (GPT) model to predict network traffic and a fine-tuned Bidirectional Encoder Representations from Transformers (BERT) to evaluate the predicted traffic. An LSTM classifier then identifies malicious packets among these predictions.

Another key research stream is the use of LLMs to discover attack procedures. To this extent, the work proposed by Ying Zhang *et al.* [12] explores the development and deployment of GenTTP, a zero-shot framework that employs LLMs to analyze and automatically generate Tactics, Techniques, and Procedures (TTPs) for interpreted malware specific to Open-Source Software (OSS) ecosystems (e.g., PyPI, NPM). Unlike traditional malware analysis, which relies on manual reverse engineering or dynamic sand-boxing, GenTTP focuses on software supply chain (SCS) attacks, analyzing malicious OSS packages designed to infiltrate systems via deceptive metadata and stealthy code execution.

Furthermore, Lingzhi Wang *et al.* [13] introduce Aurora, a semi-automated Breach and Attack Simulation (BAS) system designed to construct multi-step cyberattack chains efficiently. Aurora leverages LLMs, such as GPT, along with the Planning Domain Definition Language (PDDL), to address challenges in traditional BAS, such as limited action space, expertise requirements, and insufficient attack chain realism. Aurora automates the simulation of full cyberattack chains by analyzing attack tool documentation, threat intelligence reports, and real-world techniques.

The goal of the current paper is to improve the defenses and resilience of systems focused on cyber defense environments. Under this paradigm, Christoforus Yoga Haryanto *et al.* [14] demonstrate how contextualized AI can accelerate strategic decision-making by conducting an automated literature survey

using LLMs. Contextualized AI significantly enhances cyber defense by accelerating threat intelligence synthesis, enabling strategic decision support, improving human-AI collaboration, and facilitating proactive defense posturing.

Also, Johannes F. Loevenich *et al.* [15] focus on the design and training of robust Autonomous Cyber Defense (ACD) agents for use in military networks. It introduces an architecture that combines a hybrid AI model incorporating Multi-Agent Reinforcement Learning (MARL), LLMs, and a rule-based system. These components are organized into blue and red agent teams distributed across network devices. The primary objective is to automate essential cybersecurity functions, including monitoring, detection, and mitigation, thereby enhancing the ability of cybersecurity professionals to safeguard critical military infrastructure.

The previous related works highlight LLMs' transformative potential in cybersecurity and cyber defense. These studies demonstrate LLMs' capabilities in strategic reasoning, IoT intrusion prediction, malware TTP analysis, enhancing BAS systems, synthesizing cybersecurity literature, and building Autonomous Cyber Defense agents. Collectively, they showcase LLMs' versatility in addressing various cybersecurity challenges, from threat detection to automated defense in diverse environments. However, while these contributions demonstrate significant advancements, a crucial gap remains in effectively leveraging LLMs to analyze comprehensive attack procedure databases and integrating this analysis into a SCE methodology. This unexplored approach could enable more sophisticated adversary behavior anticipation, allowing for the proactive design of adaptive security mechanisms that anticipate threats before they materialize, bridging critical gaps in the continuous development lifecycle.

III. Large Language Model as an enabler of adversary anticipation

This section presents the proposed flow for generating attack-defense trees that represent potential adversary behavior, as well as the key metrics defined to evaluate the quality of the trees generated by LLMs.

A. Generating an attack-defense tree

Attack-defense trees provide a structured framework for modeling and analyzing adversarial interactions in cybersecurity scenarios by integrating both offensive and defensive actions into a single hierarchical representation. These trees extend traditional attack trees by incorporating defensive countermeasures, enabling analysts to visualize the dynamic interplay between attackers seeking to exploit vulnerabilities and defenders implementing mitigation. The visual nature of attack-defense trees also enhances interdisciplinary collaboration, enabling technical teams and stakeholders to jointly evaluate security postures and prioritize mitigation against multi-stage threats.

Consequently, in our proposal, the attack-defense trees are used as part of a cyber defense strategy, allowing an organization that uses such trees to evaluate its own critical vulnerabilities and predict adversary behavior patterns according to known attack procedures. As a result, such an organization can pose defense strategies that are adequate considering a cost-benefit analysis and the capacities of the adversary. For the generation of the attack-defense trees, we have provided the LLMs with structured, individual prompts focused on single tasks, rather than a single prompt containing multiple tasks, which offers significant advantages.

LLMs tend to perform better when solving smaller, discrete tasks compared to handling multiple complex tasks simultaneously. This phenomenon can be attributed to the model's inference capabilities, which are constrained by the number of tokens it can process. Another benefit of using multiple individual prompts lies in the model's recall capability. This is particularly important as it enables the LLM to infer attacks and controls that are more contextually aligned with the specific system or infrastructure components being analyzed.

Thus, we propose the flow diagram depicted in Figure 1, which represents the interactions that a cybersecurity analyst should have with an LLM to construct an attack-defense tree. Such a flow will be described next and is composed of the following 6 phases: *Application Security Context, Prompt Context, Insert Prompt, Attack Context, Cosmetic Context, and Expert validation.*

1) Phase 1: Application Security Context: The flow depicted in Figure 1 begins by establishing the context for the LLM model by introducing key concepts such as application security, threat modeling, and attack-defense trees. This step is crucial to ensure that the model remains focused on the relevant subject matter for future tasks and avoids generating content unrelated to these topics. To achieve this, it is possible to pose questions to the LLM about these areas and request summaries of the information.

This approach helps consolidate concise and relevant knowledge within the model's memory, which is inherently limited. Questions like "What is application security?", "What is threat modeling?", and "What is threat modeling using attack trees?" serve as effective starting points.

2) Phase 2: Prompt Context: In the following phase, the LLM model is provided with structured prompts and generates the desired output in the form of an attack-defense tree using DOT format, which is a way to represent these trees as directed graphs using the Graphviz language [16]. The structured prompt format defined for this approach is as follows:

- **System Context**: The context of the system is provided in a specific form, e.g., "The context of the system will be provided by critical mission objectives and operational threat scenarios".
- **Component List**: It contains a list of technologies that will be used to deploy the system, along with the corresponding safeguards for each technology, e.g., "AWS EC2 (monitored by Amazon GuardDuty to detect anomalous accesses)".
- **Attack Goals**: It refers to a phrase that indicates to the LLM that an attack goals is provided, e.g. "Ex-filtrate supply chain information".

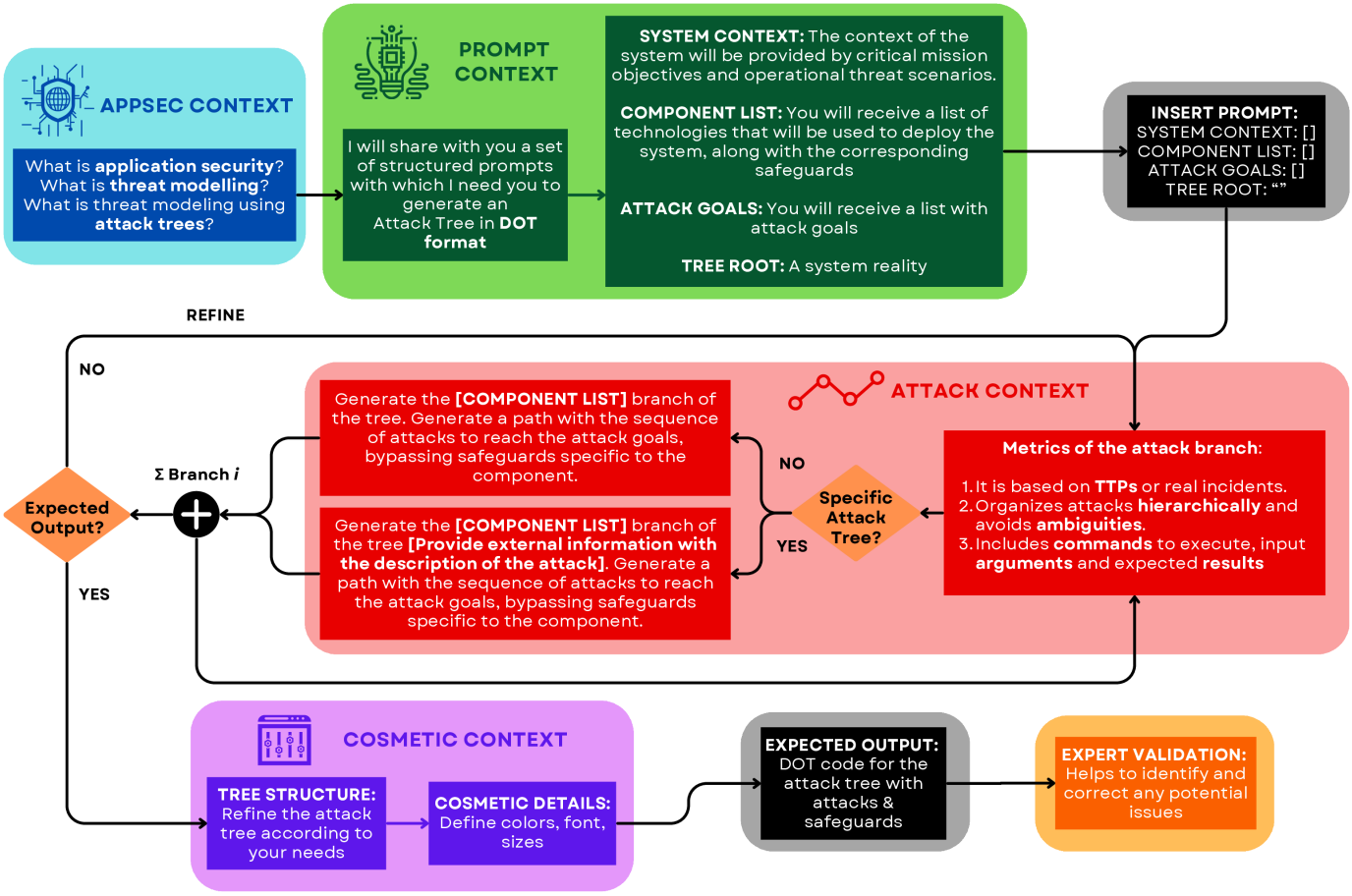

Figure 1: Flow diagram of the construction of an attack-defense tree using LLM

- **Tree Root**: The main node of the tree is specified to the LLM, e.g., "Cloud-based supply chain System".

The decision to use a 4-parameter prompt was based on a series of tests, which revealed that including additional parameters led LLMs to produce meaningless code. It is important to note that the order of the parameters had no impact on the outcomes.

3) Phase 3: Insert Prompt: This phase is critical to the process as it requires summarizing all the knowledge about the system under development. The outcomes at this stage can vary depending on the LLM being used.

4) Phase 4: Attack Context: In order to achieve certain quality in the results produced by LLMs, it becomes necessary to establish a series of metrics. After reading the literature, we can determine that an attack branch should be realistic, meaning it should be based on documented TTPs (Tactics, Techniques, and Procedures) from frameworks such as MITRE ATT&CK or real-world incidents. Additionally, the steps should be organized hierarchically (objectives, steps, and commands) and avoid ambiguities. Finally, the branches should include executable commands, adjustable parameters, e.g., IP addresses, resource names, among others, and expected outcomes.

Thanks to the versatility of the LLM models, it is possible to generate both generalized and specific attack-defense branches. On one hand, generalized attack-defense branches offer a broad overview of possible attacks a system might face. However, it requires a security analyst's expertise to determine whether the inferred attacks are genuinely exploitable within the system. To create a generalized attack-defense branch, the security analyst must prompt the LLM to infer potential attacks based on the system's context, topological components, safeguards, and attack goals.

On the other hand, a specific attack-defense branch provides a more detailed view of the exact sequence of attacks an adversary might follow to achieve their objectives. This approach is particularly useful for security analysts who need precise attack procedures. To create a specific branch, the security analyst must provide the LLM with resources containing a detailed attack, such as one of the attack procedures included in HackTricks [17].

For better results, the security analyst can ask the LLM to generate branches independently for each system component and then combine them into a complete tree. Afterward, it is essential to verify whether the generated attacks align with the intended attack goals. If not, those branches should be refined by iterating the process and including the details needed inside the prompts.

5) Phase 5: Cosmetic Context: Once the security analyst is satisfied with the generated attack-defense tree, he/she can request the LLM to apply cosmetic adjustments, such as modifying the text font, color, size, or node colors, among other visual elements. Additionally, the analyst can instruct the LLM to redefine the tree's structure if necessary. This process may be required in cases where the generated attack-defense tree contains redundant connections between nodes or if the arrangement of parent nodes or attack objectives does not meet the analyst's expectations.

6) Phase 6: Expert Validation: Once the attack-defense tree is generated, it undergoes a thorough validation by a security analyst. If the output does not meet expectations, e.g., if it includes irrelevant attacks, fabricated controls, disconnected nodes, or lacks meaningful relationships between attacks and goals, the analyst can provide feedback. The tree can be returned to the LLM in DOT format with specific instructions for revisions. This expert validation process is crucial for identifying and addressing potential shortcomings, ensuring that the final output is accurate, reliable, and applicable to real-world scenarios.

B. Measuring the quality of an attack-defense tree

In this proposal, these attack-defense trees are intended to support the construction of SCE experiments, and to achieve this objective, they must have a well-defined structure and avoid ambiguities that could delay or even prevent the subsequent validation of attack scenarios aimed at anticipating adversary behavior. To this end, the following metrics are proposed: Attacks based on known TTPs, Ordered attack procedures, and Usable attack procedures.

Attack-defense trees benefit from incorporating the first one as this grounds the model in real-world adversary behaviors, ensuring scenarios align with documented threat intelligence and reducing ambiguity during validation. Ordered attack procedures enforce logical sequences of attack steps, reflecting the dependencies and progression of real-world attacks, which is critical for structuring reproducible experiments and accurately anticipating adversarial workflows. Finally, usable attack procedures prioritize actionable, non-redundant steps that are feasible to implement in controlled experiments, avoiding overly complex or theoretical paths that could hinder practical validation. Together, these metrics enhance the clarity, realism, and operational relevance of attack-defense, enabling systematic testing of defenses against validated attack vectors while minimizing ambiguities in scenario execution.

1) Attacks based on known TTPs: This metric ensures that the tree incorporates real TTPs used in established frameworks such as MITRE ATT&CK or the Cyber Kill Chain framework, which are already well-structured. Thus, minimizing potential hallucinations by the models when generating attack paths.

$$MITRE_{Score} = \frac{\sum_{i=1}^{n} M_i}{n} \times 100 \tag{1}$$

- M_i = Binary value (1 if node i corresponds to a MITRE ATT&CK technique appropriate to the attack being carried out, 0 otherwise)
- n = Total number of attack nodes in the tree

2) Ordered attack procedures: The branches of the tree must be organized hierarchically, starting from a root node that connects the attack nodes and ends in a target node. Additionally, attack paths must be clear and unambiguous, accurately reflecting the process an attacker follows to exploit a vulnerability:

$$Ordered_{Score} = (1 - \frac{N_d + N_{sc}}{n}) \times 100 \tag{2}$$

- N_d = Number of nodes deviated from the original order established in the external source attached to the LLM
- N_{sc} = Number of nodes without children that are not a final node
- n = Total number of attack nodes in the tree

3) Usable attack procedures: Includes commands to execute, input arguments and expected results. Attack and defense nodes must clearly describe what is done (procedure), how it is done (commands and input arguments), and the expected outcome:

$$Usable_{Score} = \frac{\sum_{i=1}^{n}(C_i + I_i + R_i)}{3n} \times 100 \tag{3}$$

- C_i = 1 if the node contains at least one real executable command appropriate to the attack being carried out, 0 otherwise.
- I_i = 1 if the node contains input parameters necessary to execute the command, 0 otherwise.
- R_i = 1 if the node contains descriptive expected results, 0 otherwise.
- n = Total number of attack nodes in the tree.

Finally, we can compute the overall score of the tree with the following equation:

$$Tree_{Score} = \frac{MITRE_{Score} + Ordered_{Score} + Usable_{Score}}{3} \tag{4}$$

IV. Experiments

This section of experiments is composed in the following way: Section IV-A where the cyber defense scenario is established, Section IV-B, where the attack-defense trees are generated, Section IV-C, where one of the branches from the previously generated attack-defense trees is selected and translated into a SCE experiment, and Section IV-D, where the results obtained from the attack generation process are presented and analyzed.

A. Settings

This section demonstrates the practical implementation of the methodology proposed in Section III through a military sector case study motivated by the critical need to model and anticipate adversarial tactics in cyber defense planning. In this case study, a Ministry of Defense (MoD) operates a military logistic information system with a cloud-native approach, designed to get the most benefits of cloud capabilities while also being resilient enough to counter possible sophisticated threat actors. This military logistics system track essential war

fighter supplies across global bases and handles classified data including ammunition, medical kits, and rations, which need to be protected against nation-state cyber espionage tactics. The military logistics system implements a security-first design philosophy which stems from historical analysis of adversaries behavior targeting military logistics systems.

With the aim of defining a specific scenario for the experiments, it is assumed that the military logistics system is deployed in AWS GovCloud and employs three key managed services: EC2 for secure computing, CodeBuild for automated CI/CD pipelines, and CodeGuru for AI-powered code security analysis. This cloud architecture provides a realistic scenario for assessing the capabilities of LLMs in generating attack-defense trees based on the data flow presented in Section III. Building upon this foundation, two specific LLMs were selected for their evaluation in reasoning and cybersecurity applications: GPT-4 and QwQ-32B. These models will serve as the core components for simulating adversary behaviors and creating structured attack-defense models, further enhancing the experiment's relevance and applicability to real-world cybersecurity scenarios. For the experiment, we have used the web versions of both models, available at https://chat.qwen.ai and https://chatgpt.com, respectively.

B. Generating Attack-Defense trees

The attack trees generated when applying the proposed iterative process with both models are shown in Figures 2 and 3. Each branch begins with a node (represented in dark blue) that identifies the specific AWS service being targeted. From there, each branch follows a series of attack nodes (represented in light blue) that detail the specific steps, commands, and techniques used in the attack sequence. Despite the diversity in attack methodologies, all three paths ultimately converge toward the same goal: executing a privilege escalation attack.

- **Branch 1:** In the first branch, i.e., EC2 attack branch, the attacker gains access to EC2 resources, executes a reverse shell to establish a connection with their systems, and steals sensitive credentials from the instance. This allows them to escalate privileges and access additional AWS resources.
- **Branch 2:** In the second branch, i.e., CodeBuild attack branch, the attacker uses valid credentials to set up a compromised CodeBuild project that runs a reverse shell, enabling them to steal credentials from the build environment. They then delete the project to cover their tracks while maintaining unauthorized access to AWS resources.
- **Branch 3:** In the third, i.e., CodeGuru attack branch, the attacker injects malicious code into a repository and manipulates configurations to expose sensitive information during automated code reviews. By extracting credentials from the analysis process, they escalate privileges and gain unauthorized access to sensitive data.

The trees generated by both LLMs were evaluated based on the three key metrics proposed in Section III-B. Overall, as we can appreciate in Table I, a comparison of final scores reveals that QWQ-32B achieved a $Tree_{Score} = 71.60\%$, while GPT-4 scored a $Tree_{Score} = 61.73\%$. In mapping real-world TTPs from the MITRE ATT&CK framework, QwQ-32B shows a more reliable representation with minimal hallucination, compared to GPT-4, which often provides incomplete or incorrect information. Both models excel in creating hierarchical structures for attack procedures, but QwQ-32B offers clearer paths and priorities. Importantly, QwQ-32B outperforms GPT-4 in providing usable attack procedures, offering more detailed commands and expected outcomes. Having all this information considered, this makes QwQ-32B the preferred choice for subsequently detailing an SCE experiment from this tree.

Metric	QwQ-32B	GPT-4
Attacks Based on Known TTPs	Reliably maps real-world TTPs from MITRE ATT&CK with minimal hallucination (22.22%) (4/18 nodes).	Most TTPs are incomplete or wrong, slightly reducing realism (11.11%) (1/9 nodes).
Ordered Attack Procedures	Clear hierarchical structure with unambiguous paths and well-defined priority (100%).	Hierarchical but occasionally introduces minor ambiguities in node relationships (100%).
Usable Attack Procedures	Highly detailed commands, inputs, parameters, and expected outcomes (92.59%) (50/18 nodes).	Provides usable steps but lacks technical depth in some areas (74.07%) (20/9 nodes).

Table I: Comparative table of obtained metrics

C. SCE experiment

The purpose of this experiment is to evaluate the resilience of an AWS environment to potential misuse of permissions. Specifically, the experiment will test whether an attacker can successfully request a Spot Instance, attach a privileged IAM role, and execute a reverse shell script to steal the IAM role credentials, using the permissions $ec2 : RequestSpotInstances$ and $iam : PassRole$.

Based on the goal of the attack tree, i.e., perform a privilege escalation attack, it is possible to define the experiment following the SCE methodology:

- **Observability**: Detection of Spot Instance creation with suspicious user data and anomalous permission usage via AWS GuardDuty.
- **Steady State**: A secure AWS environment where GuardDuty detects privilege escalation, misconfigured permissions are non-exploitable, and security controls function as designed.
- **Hypothesis**: AWS GuardDuty will detect privilege escalation attempts when Spot Instances with misconfigured permissions and malicious user data are launched.

D. Experiment Methodology

This section describes the execution of the SCE Privilege Escalation experiment using a Spot EC2 instance. The automation of the experiment, which replicates the attack from branch #1 of the attack tree shown in Figure 3, culminates in obtaining temporary credentials for the $EC2-CloudWatch-$

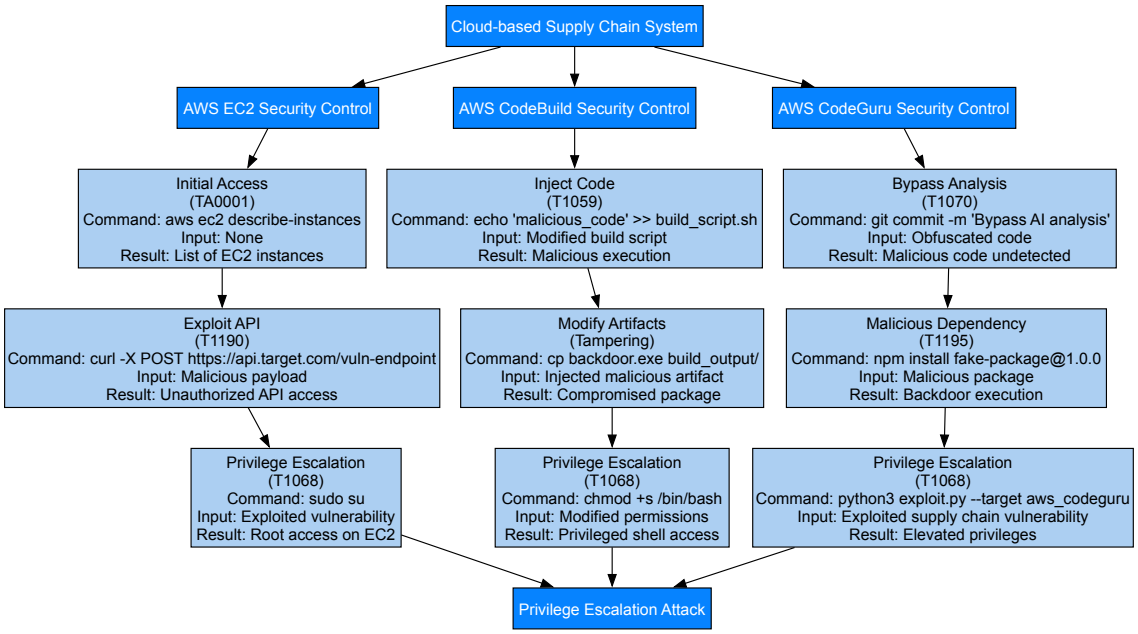

Figure 2: Reduced version of the attack-defense tree generated by GPT-4

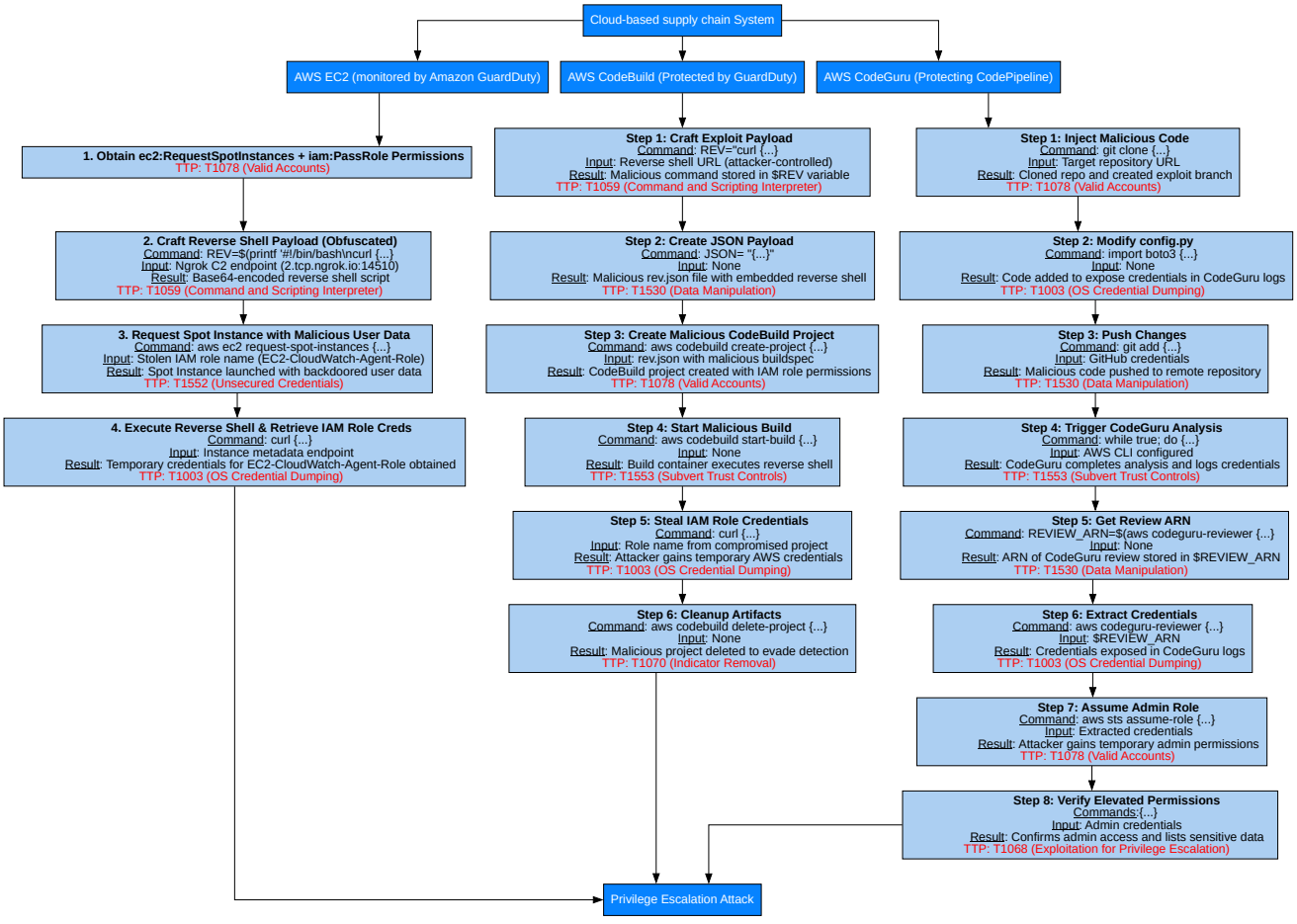

Figure 3: Reduced version of the attack-defense tree generated by QwQ-32B

$Agent - Role$. Further detailed information about the experiment, along with the steps to replicate it, can be found in the following repository: https://github.com/mariomc14/devsecops-adversary-llm.git. Specific actions performed per each stage of the experiment are described next:

- GuardDuty Findings: The initial state of Amazon GuardDuty is verified, ensuring that no prior alerts exist before executing the subsequent phases of the experiment. If no alerts are detected, the execution proceeds with the creation of the Spot EC2 instance.
- Spot EC2 Instance Creation: The command to create the instance with the $EC2 - CloudWatch - Agent - Role$ is executed, incorporating the Base64-encoded string as a payload in the UserData field. During the instance creation process, a connection for reverse shell listening is established in parallel.
- Credential Extraction and Usage: In the third and final stage of the experiment, the obtained credentials are extracted and used by executing the request shown below. These credentials are stored and leveraged to perform actions that the attacker initially did not have permission for, such as listing users.

V. Conclusions and Future Work

In this paper, we proposed an LLM-based flow for generating attack-defense trees that represent adversary behavior. We defined key metrics to evaluate the quality of the generated attack-defense trees. Based on the generated diagrams, we constructed an SCE experiment, relying on one of the previously selected branches of the attack-defense tree. Subsequently, we executed the SCE experiment in ChaosXploit, achieving privilege escalation in an EC2 Spot instance, as part of a military supply chain system in a cyber defense scenario. The results demonstrated the capability of the attack-defense trees generated by our proposed flow to anticipate adversary behavior in exploiting pre-existing system vulnerabilities, and its utility in a cyber defense strategy.

Future work will explore extending this framework to support automated countermeasure recommendations to enhance its practicality for anticipating adversary behavior in real-world scenarios.

Acknowledgment

This work has been supported by ECYSAP EYE, which is a EDF program funded by the EC under the GA Project 101168092. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them. This work has also been partially supported by MCIN/AEI/10.13039/501100011033 NextGeneration EU/PRTR, UE, under Grant TED2021-129300B-I00, by MCIN/AEI/10.13039/501100011033/FEDER, UE, under Grant PID2021-122466OB-I00, by the Spanish National Institute of Cybersecurity (INCIBE) by the Recovery, Transformation and Resilience Plan, Next Generation EU under the strategic project DEFENDER, by the CyberDataLab (Cybersecurity and Data Science Laboratory) at the University of Murcia (Spain), and the School of Engineering, Science and Technology at the University of Rosario (Colombia).

References

[1] J. Srinivas, A. K. Das, and N. Kumar, "Government regulations in cyber security: Framework, standards and recommendations," *Future Generation Computer Systems*, vol. 92, pp. 178–188, 2019.

[2] C. Serban, A. Baluta, D. Bucerzan, D. Gavrilut, A. Sobolev, and S. Oprea, "Malware classification meets vision transformation: A survey," *arXiv preprint arXiv:2304.01142*, 2023.

[3] K. A. Torkura, M. I. H. Sukmana, F. Cheng, and C. Meinel, "Cloudstrike: Chaos engineering for security and resiliency in cloud infrastructure," *IEEE Access*, vol. 8, pp. 123 044–123 060, 2020.

[4] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "Devops," *IEEE Software*, vol. 33, no. 3, pp. 94–100, 2016.

[5] A. Silva, S. Fang, and M. Monperrus, "RepairLLaMA: Efficient representations and fine-tuned adapters for program repair," *arXiv preprint arXiv:2312.15698*, 2023. [Online]. Available: https://arxiv.org/abs/2312.15698

[6] J. Zhang, H. Bu, H. Wen, Y. Liu, H. Fei, R. Xi, L. Li, Y. Yang, H. Zhu, and D. Meng, "When llms meet cybersecurity: A systematic literature review," *arXiv preprint arXiv:2405.03644*, 2024. [Online]. Available: http://arxiv.org/abs/2405.03644

[7] OpenAI, "Chatgpt: Large language model for conversational ai," https://openai.com/chatgpt, 2024, accessed: 2024-03-19.

[8] J. Yang, H. Jin, R. Tang, X. Han, Q. Feng, H. Jiang, S. Zhong, B. Yin, and X. Hu, "Harnessing the power of llms in practice: A survey on chatgpt and beyond," *ACM Transactions on Knowledge Discovery from Data*, vol. 18, no. 6, p. Article 160, 2024. [Online]. Available: https://dl.acm.org/doi/10.1145/3649506

[9] M. Bedoya, S. Palacios, D. Díaz-López, E. Laverde, and P. Nespoli, "Enhancing devsecops practice with large language models and security chaos engineering," *International Journal of Information Security*, vol. 23, no. 6, pp. 3765–3788, 2024. [Online]. Available: https://link.springer.com/article/10.1007/s10207-024-00909-w

[10] Y. Zhang, S. Mao, T. Ge, X. Wang, A. de Wynter, Y. Xia, W. Wu, T. Song, M. Lan, and F. Wei, "Llm as a mastermind: A survey of strategic reasoning with large language models," *arXiv preprint arXiv:2404.01230*, 2024.

[11] A. Diaf, A. A. Korba, N. E. Karabadji, and Y. Ghamri-Doudane, "Beyond detection: Leveraging large language models for cyber attack prediction in iot networks," in *2024 20th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*. IEEE, 2024, pp. 117–123.

[12] Y. Zhang, X. Zhou, H. Wen, W. Niu, J. Liu, H. Wang, and Q. Li, "Tactics, techniques, and procedures (ttps) in interpreted malware: A zero-shot generation with large language models," *arXiv preprint arXiv:2407.08532*, 2024.

[13] L. Wang, Z. Li, Z. Guo, Y. Jiang, K. Jung, K. Thiagarajan, J. Wang, Z. Wang, E. Wei, X. Shen, and Y. Chen, "From sands to mansions: Simulating full attack chain with llm-organized knowledge," 2024. [Online]. Available: https://arxiv.org/abs/2407.16928

[14] C. Y. Haryanto, A. M. Elvira, T. D. Nguyen, M. H. Vu, Y. Hartanto, E. Lomempow, and A. Arakala, "Contextualized ai for cyber defense: An automated survey using llms," *arXiv preprint arXiv:2409.13524*, 2024.

[15] J. F. Loevenich, E. Adler, A. Bécue, A. Velazquez, K. Wrona, V. Boshnakov, J. Falkcrona, N. Nordbotten, O. L. Worthington, J. Röning, R. Rigolin, and F. Lopes, "Training autonomous cyber defense agents: Challenges & opportunities in military networks," in *MILCOM 2024 - 2024 IEEE Military Communications Conference (MILCOM)*, 2024, pp. 158–163.

[16] "Graphviz," accessed: May 5, 2025. [Online]. Available: https://graphviz.org

[17] "Hacktricks," accessed: March 10, 2025. [Online]. Available: https://book.hacktricks.wiki/en/index.html