Securing software development lifecycle using Artificial Intelligence and Security Chaos Engineering

Author
Martín Steven Bedoya Rodríguez

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

Director
PhD. Daniel Díaz-López

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

Bogotá - Colombia
2024

Acknowledgment

To God for giving me life and guiding me on this path.

To my parents, for giving me the tools to achieve this goal.

To my beloved wife Laura for her inspiration, support and patience.

To my son, for letting me know true love.

To my sister, for her English lessons.

To Professor Daniel for sharing his knowledge and wisdom with me.

To all those who were part of this process by giving their grain of sand.

Infinite thanks.

Abstract

Although software development processes have been substantially optimized in recent years, security breaches continue to represent a major risk factor for companies. Digital transformation processes that do not implement security as a pillar tend to generate reprocesses and additional costs for remediation, in the worst cases, information leaks, ransomware attacks, denials of service, and other cyber-attacks that generate strong reputational, legal, or monetary impact. Over the years, secure development practices have evolved, and today frameworks help to intrinsically prevent vulnerabilities. Application Security Programs have also emerged, which are a mechanism that encompasses the policies, guidelines, processes, tools, and people that companies implement to protect their applications. Increasing the maturity level of an application security program requires automating activities and defining novel ways to challenge application security. One of the mechanisms in ague to automate security activities is Artificial Intelligence, with the advent of Large Language Models it is possible to solve tasks in each phase of the software life cycle, reducing the time spent by companies to generate secure systems. On the other hand, through Security Chaos Engineering it is possible to discover new forms of risk that are not easily discovered through traditional pen-testing methods or automated tools, which improves the security posture of applications. This master thesis generates a series of contributions that allow companies to improve their Application Security Programs. This work introduces ideas on early threat identification by applying Natural Language Processing on user stories, demonstrates the automation of threat models based on attack-defense trees using Large Language Models, and proposes Security Chaos Engineering use cases applicable to DevSecOps practices.

Contents

0.1 Introduction

The software development process has evolved rapidly in recent decades, agile methodologies and DevOps practices have allowed to build software quickly, increasing profits for software factories and accelerating the digital transformation of companies. Historically, the need to create software quickly has gone against secure systems, since development teams focus on generating value quickly and security tasks are not a priority. Sometimes security is only applied in the testing phase, through penetration tests, but this generates reprocesses and delays in the transition to production since the development team must remediate the vulnerabilities detected without affecting the functionality, user experience, or performance of the system.

Securing the software lifecycle has become a fundamental task to guarantee secure systems. Regulatory frameworks such as SOX [1], HIPAA [2] or PCI [3], oblige development teams to secure software and guarantee that the systems developed have a minimum criterion of confidentiality, integrity, and availability of user data. Ensuring the construction of secure software requires the implementation of an Application Security Program [4]. An application security program is a set of policies, guidelines, tools, and people that work in harmony in order to secure the lifecycle of all software that is acquired and produced in an enterprise. Governance frameworks such as SAMM [5] can be used to define policies and guidelines, which specify the security activities to be performed at each phase of the software lifecycle. In addition, investing in security tools and trained personnel is necessary to operate the program.

When a company does not implement an Application Security Program, it exposes itself to insecure systems. Even if a company does not have a software in-house factory, it should require its software vendors to implement a minimum of security, otherwise, it is exposed to multiple reputation or legal impacts that can even lead to bankruptcy. For example, according to CISQ, the annual cost of cybercrime increased from $3 trillion in 2015 to $6 trillion in 2021, and the cost of vulnerability-related flaws in web applications rose to $2.2 trillion [6]. Thus, we may affirm companies should choose an Application Security Program that fits their business needs and reduces the risk of cyber fraud.

An Application Security Program may have different security enablers, which are key requirements for approving a software release to production, for example: requiring a security architecture, a threat model, a code analysis, and ethical hacking can be examples of security enablers. The number of enablers will depend on the maturity level of the application security program and will be a point of optimization of the program.

One of the most common security enablers in an Application Security Program is the threat model. Early identification of security threats reduces the cost of vulnerability remediation, as developers can apply strategies to avoid security issues directly in the source code or architects can propose a perimeter protection. However, building a threat model is a practice that can be costly, as it requires experienced cybersecurity professionals who understand the needs of the business.

To begin with, when a software factory implements an Application Security Program, threat modeling is expected to be an activity performed by the development team, usually, these teams lack the technical expertise to explore the tactics, techniques, and procedures needed to affect the software, that is why usually threat modeling is an activity performed by the companies' cybersecurity team.

There are several mechanisms to generate a threat model, and each one has advantages and disadvantages, in all cases, it is necessary to have an important knowledge of the business. Nowadays the construction of a threat model must be done in an agile way, this implies making threat models by sprints, anticipating the threats that may affect the future functionality. One interesting mechanism to generate threat models that adapt to agile environments is the attack-defense trees since the branches of the tree can be organized during

the different sprints of the development.

With the advent of Large Language Models (LLMs), which is a branch of artificial intelligence, it is possible to automate the threat modeling process, these LLMs are trained with huge amounts of information including cybersecurity, cyber-threat, and cyber-fraud aspects. Through LLMs it is possible for a cybersecurity professional to reduce the time it takes to perform threat modeling, as LLMs can infer attacks from a given context, this can help reduce the cost of this security enabler. This project demonstrates the use of LLMs to generate attack-defense trees to quickly identify the techniques that adversaries can use to affect the software.

Another common security enabler is security testing, it is usually required that the software does not have critical or high vulnerabilities to go into production. For this purpose, companies can implement Static Application Security Testing (SAST) or Dynamic Application Security Testing (DAST). These two security enablers will be explored in this master thesis.

On the other hand, concerning the identification of vulnerabilities when the software is in the coding and testing phase, there are a large number of security tools, such as ZAP [7], BurpSuite [8], Snyk [9], Fortify [10], which allow automating the discovery of vulnerabilities in the source code, in third-party libraries and the running functionality. With the advent of the DevSecOps practice, the vulnerability identification process has been substantially optimized in cost and time.

In a DevSecOps practice, all stakeholders in the software process perform their specific work leaving the CI/CD pipelines to perform the tasks of building and deploying new features. This is where SAST and DAST tools can offer an advantage by quickly scanning the software for vulnerabilities. However, there is an over-reliance on the toolchain, making it possible to implement vulnerabilities when security tools are not robust enough. This is where the power of Security Chaos Engineering comes in, which is a novel way to challenge the security of systems in order to identify vulnerabilities that are not easy to detect.

Security Chaos Engineering offers a mechanism for identifying vulnerabilities through proactive experimentation [11]. SCE uses the scientific method to discover and measure new forms of risk that cannot be detected with automatic tools or penetration tests. For example, through SCE it is possible to determine how a software functionality would behave when a developer alters system roles, or it would also be possible to identify if an infrastructure component privilege opens a backdoor.

Thus, in this master thesis we demonstrate how Security Chaos Engineering strengthens a DevSecOps practice because, through this novel method, it is possible for cybersecurity experts to devise new forms of risk. In contrast, security tools identify traditional vulnerabilities like the ones included in OWASP TOP10 [12] or SANS/CWE25 [13]. We also propose a new way to secure the software development lifecycle by integrating Large Language Models (LLMs) to discover and test the tactics, techniques, and procedures employed by adversaries to compromise software.

The remainder of this document is structured as follows. Section 0.2 describes the general objective and the specific objectives of this master thesis, being artificial intelligence and Security Chaos Engineering as the main project's cornerstones. In Section 0.3, the research process followed during the two semesters of the year 2023 is detailed, it should be noted that it was a continuous learning process in accordance with the courses seen in the master's degree that involved the applications of artificial intelligence in cybersecurity. Then, Section 0.4 describes the artifacts generated as a result of this research, mainly a set of talks and papers submitted to different conferences and journals. Finally, Section 0.5 concludes the master's thesis, presenting some future research paths, both in artificial intelligence and Security Chaos Engineering.

0.2 Objectives

General Objective

Propose methods to identify threats early and determine attack strategies using Artificial Intelligence and Security Chaos Engineering to secure the software lifecycle.

Specific objectives

- OBJ1: Propose a mechanism to identify attack strategies from a set of user stories and test them through SCE methodology.
- OBJ2: Propose a solution that integrates a NER model and an attack-defense tree to contribute to securing software in military environments.
- OBJ3: Propose an innovative mechanism involving LLMs and SCE to strengthen the maturity level of a DevSecOps practice.
- OBJ4: Demonstrate the applicability of SCE in agile software construction environments, from the identification of TTPs and efforts on the attacker's side.

0.3 Methodology

The methodological process followed to achieve the previously mentioned master thesis objectives is described next.

0.3.1 First semester

In the first academic semester of 2023 (January - May) we sought to apply artificial intelligence in a cybersecurity environment related to Application Security. Since artificial intelligence is a broad field, natural language processing was chosen due to the experience in securing software life cycle, since almost always the construction of a system involves a series of activities that are documented in natural language.

One of the objectives was to identify how Natural Language Processing could support the assurance of the software life cycle and for this, a review of the state of the art was made by looking for different models of natural language processing that were applied in security activities during the construction of a software, then there were identified a set of research and published works in the different phases of the life cycle, works were found in the phases of analysis, design, testing and deployment. This allowed the conclusion that there are several natural language processing models applicable to secure the different phases of the software life cycle, it is an idea that was introduced some years ago and what was sought was to canalize these ideas to solve certain specific security tasks related to threat modeling.

This idea began with the selection of an existing natural language processing model published online that could be adapted to the context of the idea so as not to have to train a model of its own. A paper designed by authors Herwanto et al [14] was identified that allowed to identification of a series of keywords in a set of user stories. The model allows us to identify the actor, an action, and words that could be related to Personally Identifiable Information (PII). Once this model was running, a set of user history was built to see how well the model performed in identifying these keywords. This idea was presented at the BSIDES Cybersecurity Conference 2023, `https://bsidesco.org/`, held in Bogotá, Colombia, and is developed in Section 0.4.1, where we show how it is possible to generate a threat model from keywords.

Subsequently, the concept of threat modeling was deepened using attack-defense trees, because the hypotheses and experiments of Security Chaos Engineering are leveraged through attack-defense trees. These attack-defense trees are a mechanism to perform threat modeling in a fast way because they allow us to discover the different tactics and techniques that attackers can use to affect the software and make it possible to increase the branches of the tree through the different sprints of an agile software development process.

The goal was to build an attack-defense tree from the keywords identified with the Natural Language Processing model since it is precisely through these trees that Security Chaos Engineering experiments are defined. This led to the concept of trying to generate attack-defense trees that had a functional perspective, but at the same time focused on tactics and techniques that were applicable in a software context.

From this idea, the first use case was built for the Cyber Situational Awareness on Military Operations workshop (CSA) of the conference on Availability, Reliability, and Security (ARES 2023) `https://2023.ares-conference.eu/`, held in Benevento, Italy. Our purpose was to contribute to Cyber Situational Awareness, considering the construction of the user stories for software in a military context. A possible solution architecture for such software was defined and the IriusRisk tool was used to analyze the threats that could affect that solution architecture. Then the objective was to take the set of user stories and from them detect those keywords to give the attack-defense tree a slightly more functional context, more oriented towards business logic, since the tree built from the information obtained from IriusRisk was completely oriented to threats in components. Details of this idea are depicted

in Section 0.4.2.

0.3.2 Second semester

In the second academic semester of 2023 (August - December), the concept of Large Language Models, which are an extension of Natural Language Processing, was explored in depth, and from there, the idea of automatically building an attack-defense tree that would allow the definition of hypotheses and experiments for Security Chaos Engineering was born.

In the first instance the free LLM released by OpenAI, ChatGPT-3.5, was tested, however, after a series of tests it was not possible to achieve this goal, since the attacks inferred by this model were too generalized and did not represent specific attack procedures that could be automated through Security Chaos Engineering. Subsequently, the ChatGPT-4 model was tested, obtaining two fundamental capabilities to achieve the objective: first, a better memory capacity, which avoided recall problems, and second, the capacity of this model to consult information on the Internet.

Extensive testing was performed in an attempt to replicate the attack-defense tree generated for the CSA workshop at the ARES conference, succeeding after a great deal of trial and error using different sequences of prompts. The result of this process was documented in the paper submitted to the journal Computer & Security, which is included in Section 0.4.3. In this way, a mechanism was built to generate attack-defense trees from a set of user stories and a solution architecture.

Finally, from exploring the ideas of building attack-defense tree-based threat models automatically using LLMs and then defining Security Chaos Engineering experiments applicable to DevSecOps, a magazine paper was constructed to elevate these ideas to a more executive, less technical audience. This led to the construction of a paper for the IEEE Internet Computing Magazine, which is presented in Section 0.4.4. It was sought to apply the concepts previously explored in a DevSecOps practice. A new use case involving the use of the AWS DevOps infrastructure was built, for which new user stories and a new solution architecture were defined, a new attack-defense tree was created using ChatGPT-4, and Security Chaos Engineering experiments applicable to a DevSecOps practice were defined.

0.4 Results

The results of this master's thesis are represented in different scientific contributions, which were presented at conferences or submitted to academic journals. The 4 main contributions from this thesis, which are associated with the objectives presented in Section 0.2, are described next:

- An initial approach for the integration between a Name Entity Recognition Model (NER) and SCE methodology to identify threats from user stories. This idea was submitted and **presented** in the form of a talk at the BSIDES Cybersecurity Conference 2023 in Bogotá, Colombia. Associated Specific Objective: **OBJ1**.
- A use case for securing cloud-based military systems integrating a Name Entity Recognition Model (NER) and SCE methodology. This idea was submitted and **published** as part of the proceedings of the CSA workshop at the ARES Conference 2023 in Benevento, Italy. Associated Specific Objective: **OBJ2**.
- A flowchart to generate a threat model based on attack-defense trees using LLMs to define SCE experiments applicable to a DevSecOps practice. This idea was submitted to the Computer & Security Journal, special issue in "DevSecOps: Advances for Secure Software Development", and is **under review**. Associated Specific Objective: **OBJ3**.
- A new use case for securing DevSecOps practice using the SCE methodology was proposed. It was written with a more executive rather than technical approach, contributing to the spread of SCE. This idea was submitted to IEEE Computing Magazine. and is **under review**. Associated Specific Objective: **OBJ4**.

Each one of the previously mentioned contributions will be presented next.

0.4.1 Talk presented at BSIDES Cybersecurity Conference

The BSIDES Cybersecurity Conference is a convention of hackers, analysts, consultants, researchers, and information security enthusiasts, where conferences related to offensive and defensive security are promoted. The BSIDES Cybersecurity 2022 conference was held in Bogota Colombia in October, more than 2000 participants were registered over 2 days and presented topics such as: Ethical hacking, blue team, threat intelligence, incident response, DevSecOps, artificial intelligence applied to cybersecurity, among others. More information about BSIDES Colombia 2023 may be found at `https://bsidesco.org/`

The talk presented at this conference is described below, and was the first approach between artificial intelligence and Security Chaos Engineering, specifically in this talk it is demonstrated that it is possible to analyze a set of user stories to identify attack strategies that can be tested using SCE. The talk was recorded and is accessible through `https://www.youtube.com/watch?v=byy-xTC7N9c`

0.4.2 Paper presented at ARES Conference

The International Conference on Availability, Reliability, and Security (ARES) brings together researchers and practitioners in the field of IT security and privacy, was launched in 20005, and serves as an important platform for exchanging, discussing, and transferring knowledge. The ARES 2023 conference was held in Benevento, Italy, and offered a series of specialized cybersecurity workshops, where topics such as Cybersecurity and Artificial Intelligence, Cybersecurity in physical systems and IoT, and Cybersituational Awareness, among others, were presented. More information about ARES 2023 may be found at `https://2023.ares-conference.eu/`

The paper presented at this conference is described below, and was a product of the refinement of ideas introduced in BSIDES Cybersecurity, this work demonstrates a use case where a natural language processing (NLP) model is applied to identify security keywords in a set of user stories to create a threat model based on attack-defense trees to generate SCE experiments.

Securing cloud-based military systems with Security Chaos Engineering and Artificial Intelligence

Martin Bedoya
School of Engineering, Science and Technology, Universidad del Rosario
Bogotá, Colombia
martin.bedoya@urosario.edu.co

Sara Palacios
School of Engineering, Science and Technology, Universidad del Rosario
Bogotá, Colombia
sara.palaciosc@urosario.edu.co

Daniel Díaz-López
School of Engineering, Science and Technology, Universidad del Rosario
Bogotá, Colombia
danielo.diaz@urosario.edu.co

Pantaleone Nespoli
Department of Information and Communications Engineering, University of Murcia
Murcia, Spain
pantaleone.nespoli@um.es

Estefania Laverde
School of Engineering, Science and Technology, Universidad del Rosario
Bogotá, Colombia
estefania.laverde@urosario.edu.co

Sebastián Suárez
School of Engineering, Science and Technology, Universidad del Rosario
Bogotá, Colombia
sebastian.suarezg@urosario.edu.co

ABSTRACT

Recently, system security represents a big challenge for many organizations, and it must be specifically handled when a system is intended to be deployed in a cloud environment. Cloud environments provide multiple security services that run over a Shared Responsibility Model that requires the participation of the cloud provider and the customer. Thus, this paper proposes an architecture based on Artificial Intelligence to support the finding of system threats and errors in an early stage and on Security Chaos Engineering methodology to reliably test the existence of such errors. This proposed architecture may help orientate better system designs and contribute to building holistic security. A particular use case is described to show how the proposal can be applied to a system that supports services for a military-related organization.

KEYWORDS

Security Chaos Engineering, Cloud computing, software security, military systems

ACM Reference Format:
Martin Bedoya, Sara Palacios, Daniel Díaz-López, Pantaleone Nespoli, Estefania Laverde, and Sebastián Suárez. 2023. Securing cloud-based military systems with Security Chaos Engineering and Artificial Intelligence. In *The 18th International Conference on Availability, Reliability and Security (ARES 2023), August 29-September 1, 2023, Benevento, Italy*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3600160.3605076

1 INTRODUCTION

It is clear that cybersecurity represents a real concern nowadays. That is, cyberattacks have increased in number and disruption, establishing cybercrime as one of the most beneficial black markets worldwide. In this alarming scenario, public and private entities invest several resources to secure their infrastructures and products in an effort to fight against those intrusions. So, it is essential to ensure that software development is secure during all its stages.

Besides, developing systems based on user stories is considered a significant breakthrough in the software engineering process. Typically written in natural language, these stories describe the actions that can be executed by a user in an application. Security in the software design phase is essential to ensure the early identification of security threats that may affect the system. In this phase, security activities are applied, which are only sometimes standardized and therefore depend on the experience of professionals and their business understanding. Nevertheless, no automated mechanism is available to identify the early attack strategies that cybercriminals can use to compromise one or more functionalities of an application.

In a Software Security Development Life Cycle (SSDLC), an organization defines a series of security activities to be executed in the different phases of the software process. In the design phase, the construction of deliverables is necessary, which serves as input for architects and programmers to implement security controls during the development and deployment phases. These deliverables apply to traditional software development methodologies, such as the waterfall or spiral, as well as agile methodologies. One of the most common artifacts in an SSDLC is threat models, which aim to identify and map threats that may affect the different modules and components of the software. The construction of a threat model involves a certain degree of subjectivity, depending on the experience of the cybersecurity analyst or their understanding of the business. This subjectivity leads to a certain degree of bias, making it possible for the analyst to ignore some threats, considering them improbable or simply due to lack of knowledge.

Recently, a novel paradigm named Security Chaos Engineering (SCE) has arisen in secure system development which is based on Chaos Engineering (CE) [3]. That is, SCE aims to identify security control failures through proactive experimentation, build confidence in a system's capability, and defend against malicious conditions in production. SCE methodology has been successfully applied to various contexts (e.g., cloud environment), demonstrating

ARES 2023, August 29-September 1, 2023, Benevento, Italy

ACM ISBN 979-8-4007-0772-8/23/08…$15.00
https://doi.org/10.1145/3600160.3605076

its capabilities and applicability, even if automatizing its workflows still poses a severe problem.

To solve the aforementioned challenges, this article proposes an SCE-powered framework to secure software development based on user stories. In particular, the framework implements a Natural Language Processing (NLP) model to extract relevant information from a set of user stories of a certain application. Based on this information, our proposal is able to model and identify threats that can affect the software and determine attack strategies that can be used by a cybercriminal at an early stage. Once those strategies are determined and validated, SCE is applied to the real system to test that the proposed security controls are effective against the identified threats. The integration of these two fields has the potential to enhance security testing by incorporating proactive experimentation and the identification of security control failures [14]. By combining artificial intelligence techniques with chaos engineering principles, researchers can explore new avenues for improving the robustness and resilience of software systems against cyber threats.

The remainder of this article is as follows. Section 2 reviews the most prominent works in the field of SCE and NLP applied to security problems. Then, Section 3 describes our proposal, detailing the different modules. Section 4 presents the experimental results of applying the framework to a cloud-based military environment. Finally, Section 5 concludes the article, presenting some interesting future research paths.

2 RELATED WORKS

Some proposals have been suggested for securing systems using SCE, while others have been presented to improve security thanks to the NLP methodology. To this extent, the most prominent ones are analyzed in the following Sections.

2.1 SCE proposals

Lately, there has been a recent shift in the chaotic methodology, with SCE emphasizing testing security controls to enhance system protection. SCE employs proactive experiments to build confidence in the system's ability to defend against potential threats and vulnerabilities. This shift acknowledges the inevitability of security failures and seeks to prioritize the evaluation of security measures.

In this sense, the open-source software ChaoSlingr represents a significant advancement in applying chaotic principles to information security [10]. This tool, designed for use on the AWS platform, offers a simplified approach to designing and executing security chaos experiments [14]. Its effectiveness has led to its adoption by several companies seeking to leverage chaotic experiments within their systems.

Besides, Torkura *et al.* proposed CloudStrike, an architecture that applies Risk-Driven Fault Injection (RDFI) for evaluating cloud security [20]. RDFI extends the CE paradigm to include cloud environments and leverages attack graphs to inject security faults. The authors tested CloudStrike on AWS and Google Cloud Platform, calculating risk values using the Common Vulnerability Scoring System (CVSS). Additionally, they used SCE methodology to assess CSBAuditor, a cloud security framework that continuously monitors cloud infrastructures to detect suspicious activities [19].

Moreover, SCE's application in improving API security is discussed in [15]. Particularly, the authors propose utilizing SCE to evaluate the security controls configuration of APIs, thereby identifying vulnerabilities early. By focusing on the OWASP top 10 critical web application security risks and automated attack detection, they advocate for SCE experiments to tackle these challenges.

Also, SCE experiments were conducted to assess the robustness of System of Systems (SoS) against potential attackers in [2]. Using the Chaos Toolkit, CE and SCE experiments were performed on a Virtual Unmanned Aerial Vehicle (VUAV), leveraging Attack Trees to model attacker actions. Five experiments were executed, measuring CPU and RAM usage as performance indicators.

The paper in [16] discusses the application of SCE from an economic viewpoint. Concretely, the author emphasizes that security Return On Investment (ROI) commonly relies on the successful prevention of attacks, while they are almost inevitable. In this sense, SCE culture helps in shifting the paradigm from avoiding the attacks to minimizing their impact through continuous and rigorous experimentation, system resilience, and better response plans.

From the SCE perspective, the proposed framework distinguishes itself from previous solutions by its versatility, being applicable in various contexts such as different cloud platforms (AWS, GCP, Azure, etc.), containers (Docker, Kubernetes, among others), and web applications. Another notable differentiating factor of our proposal is its high level of automation. We provide a predefined list of actions that can be sequentially executed when launching an experiment, facilitating the process and reducing manual effort.

It is worth remarking that, in our previous research, we introduced ChaosXploit [12], an innovative framework empowered by SCE principles. Building upon the research conducted in [11], ChaosXploit leverages a comprehensive knowledge database consisting of attack trees to identify and expose vulnerabilities in a certain system or software. Utilizing ChaosXploit can form an integral component of a proactive defensive security strategy, enabling the timely detection and rectification of software misconfigurations.

2.2 NLP proposals

Regarding the automation of security activities during the software development life cycle, the industry has focused on developing static analysis (Static Application Security Testing, SAST) and dynamic analysis (Dynamic Application Security Testing, DAST) tools to identify potential application vulnerabilities. Besides, there has been a focus on intrusion detection tools such as Web Application Firewalls (WAFs). By leveraging techniques in artificial intelligence and deep learning, these tools have significantly improved their detection rates and reduced false positives [7].

Currently, no commercial tools are available that automate identifying threats and attack strategies that can impact applications. However, studies have explored the use of NLP techniques to automate security activities. For instance, researchers have developed pre-trained models in the cybersecurity context to extract relevant information from bulletins, journals, and cybersecurity websites [5]. In detail, they described an ML-based NLP model designed to perform cognitive analysis on cybersecurity-related documents. The construction of a domain ontology was a key aspect of this research, achieved through a two-step approach, that is, i) the symmetry stage

and ii) the machine adjustment. Remarkably, the symmetry stage involved modeling the cybersecurity domain based on the expertise of cybersecurity professionals. A comprehensive dictionary of relevant entities was created, which was subsequently categorized into 29 classes and incorporated into the NLP model. To optimize the ontology's performance, iterative tests were conducted. Leveraging this ontology, a supervised learning-based NLP model was defined and trained using datasets consisting of approximately 300,000 words. The proposed model achieved an F1 score of 0.81 for Named Entity Recognition (NER) and 0.58 for relation extraction.

Another application of natural language processing in application security is the identification of vulnerabilities in source code, similar to what a SAST tool would do [17]. Specifically, the authors proposed a system that tackles software vulnerability detection challenges as a Natural Language Processing (NLP) problem, treating source code as textual data. The research focuses on automating the process of software vulnerability detection using state-of-the-art deep-learning NLP models. To demonstrate the feasibility of the proposal, multiple deep-learning models have been developed and compared based on their accuracy, and the highest-performing model achieved an impressive accuracy of 95%. Additionally, the study explores predicting the vulnerability class to which a given source code belongs. Then, to facilitate the practical application of the proposed system, a robust dashboard has been developed using FastAPI and ReactJS, providing a user-friendly interface for vulnerability analysis and management. The combination of advanced NLP techniques, deep learning models, and an intuitive dashboard contributes to the advancement of automated software vulnerability detection and supports effective decision-making in enhancing software security.

In other cases, NLP models are used to generate test cases from security requirements [9]. In particular, the proposal tries to addresse the challenge of automating the generation of executable security test cases from security requirements expressed in Natural Language (NL). The authors proposed, applied, and evaluated Misuse Case Programming (MCP), an innovative approach that facilitates the automatic generation of security test cases from misuse case specifications (considered of great value), which capture the behavior of malicious users. Concretely, MCP leverages NLP techniques to extract relevant concepts, such as inputs and activities, from the requirements specifications. By matching these extracted concepts with the members of a provided test driver API, MCP generates executable test cases. To assess the effectiveness of the presented approach, an extensive evaluation was conducted in an industrial case study.

These NLP-based approaches show promise in automating security-related tasks in the field of cybersecurity. Nonetheless, no study presently combines artificial intelligence with chaos engineering for cybersecurity.

3 PROPOSAL

This section describes our proposal that aims to protect cloud-based military systems using SCE and Artificial Intelligence. Figure 1 depicts the main modules of the presented framework that will be described next.

3.1 Functional Entities Recognizer

User stories are essential to design a system from a functional perspective, as these allow to comprehend the diversity of users that will interact with the system, the type of actions that the system should support and also the type of data that the system will process. Without even start implementing a system is possible to estimate many components of a new project just reviewing in detail the user stories.

From a cybersecurity perspective is also possible to analyze user stories to identify requirements in terms of roles, permissions, privacy, among other security features. But also, from user stories is possible to anticipate threats that may impact a system like information disclosure, identity impersonation, privilege elevation, among others forbidding factors.

Thus, the functional entities recognizer module has as purpose to receive a set of user stories and to identify functional entities that are at the same time relevant for a subsequent definition of a threat model. The identified entities may be of different types, even though three main entities are considered: subject, action, resource.

The detection of entities may be achieved through different ways, being one of the most accurate and automatic the use of a Natural Language Processing (NLP) model devoted to name entity recognition. NLP has been used previously in another cybersecurity contexts to identify relevant entities, suspicious accounts and anomalous activities [13, 18]. In this particular case, a NER entity recognition model focused on the detection of PII (Personal Identifiable Information) is used as part of this module.

3.2 Threat Modeler

A threat model is an important input to design systems with a security perspective, as it offers an identification of vulnerabilities and threats that could impact each component of the system according to the connections between components, the component location inside a topology and the component role.

Thus, threat models for cloud-based systems are essential as they allow to map threats specific for a cloud environment, considering factors like the types of managed services that are part of the system, the type of business model being consumed, e.g. SaaS, PaaS, IaaS, the shared responsibility model, among others.

Thus, the threat modeler module has the capability of receiving a topology of infrastructure components that integrate the cloud-based system and generate a set of threats and countermeasures.

3.3 Attack Tree Compositor

The attack tree contains routes composed of attack procedures and countermeasures, which illustrate the different alternatives that an attacker has to pursue an attack goal. The attack tree aims to offer an overview of how offensive and defensive components converge in a time line.

Thus, the attack tree compositor module integrates the outputs coming from the threat modeler 3.2 and from the functional entities recognizer 3.1 to compose an attack tree that is specific for a cloud-based military system. An example of an attack tree may be seen in Figure 3, where the light-blue boxes represent offensive actions done by an attacker, and the dark-blue boxes represent defensive actions, i.e. countermeasures, deployed by the system owner.

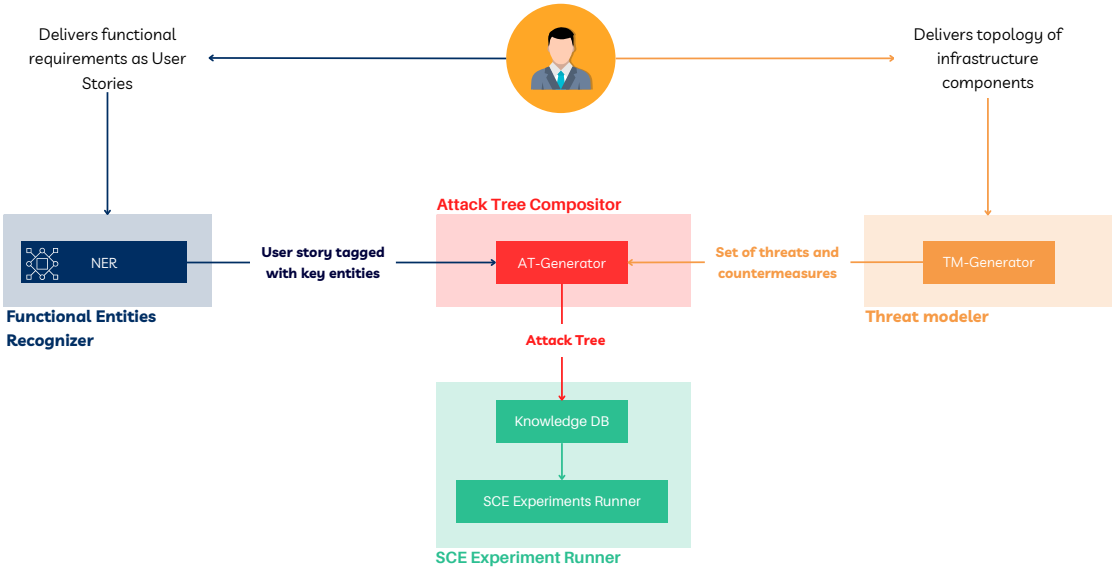


Figure 1: Proposed architecture

3.3.1 Evaluation of defensive capabilities deployed from the system. The attack tree offers a great opportunity to validate the defensive capabilities of a system if we pose an hypothetical scenario where all the offensive actions (light-blue boxes) in an attack tree, like the one shown in Figure 3, are possible and successful. It is an assumption that may be considered exaggerated and unlikely in regular scenarios, however military systems are an special case where the sensitivity of the data being managed by the system suggests that a very persistent and motivated threat actor may overpass the probabilities.

Thus, in this scenario the security test will be focused on validating the countermeasures (dark-blue) boxes that exist in the attack tree. The countermeasures are determined by the Threat Modeler and are essentially security settings over existing services or implementation of new security services in the cloud environment.

3.4 SCE Experiment Runner

Validation of security through chaos engineering, i.e. security chaos engineering, allows to discover faults in the systems that would not be discovered by traditional ways. In this way, the steps of a chaos engineering methodology represented by observe what to measure, define a steady state, pose an hypothesis and an execute experiment, may be now applied in a cybersecurity context.

So, the SCE experiment runner receives the output of the Attack Tree Compositor (Section 3.3) and perform different security tests supported by a chaos engineering methodology, to validate the security of the cloud-based military system. Thus, this module is implemented using ChaosXploit, an open source framework that may be customized to design chaos engineering experiments and obtain a security overview from a system.

4 USE CASE: SECURING A CLOUD-BASED MILITARY SYSTEM

This section describes how the proposal introduced in Section 3 is applied in an use case in the military sector.

An organization belonging to the military sector decided to deploy some of its systems in the AWS cloud developed for DoD customer, i.e. any agency under the Department of Defense or any prime contractor. As part of this initiative, this organization decided to make a cloud-native design so all the architecture can be aligned with a cloud ecosystem. One of such systems is particularly critical as it contains information about the supply chain of essential resources for war-fighters. This system is accessed by different logistic operators deployed in military facilities who register the existence or lack of supplies. Some user stories were pose from the previous case and are represented in Listing 1.

```
1. As a logistic officer, I would like to report
    information about consumed, leftover and forecasted
    resources.
2. As a supplier manager, I would like to get information
     about delivered resources to different locations.
3. As a commanding officer, I want to be able to generate
     reports to provide visibility about supply chain
    performance.
```

Listing 1: User Stories

Considering the previous user stories, a topology of this system can be seen in Figure 2. This topology contains 2 ways of access represented by a web browser and a mobile application. Additionally, the main components that support this system are: i) an ECS container and a RDS database that supports a secret authentication process, ii) an EC2 instance (web server) and a DynamoDB database that handle the information provided by logistic officers, iii) a

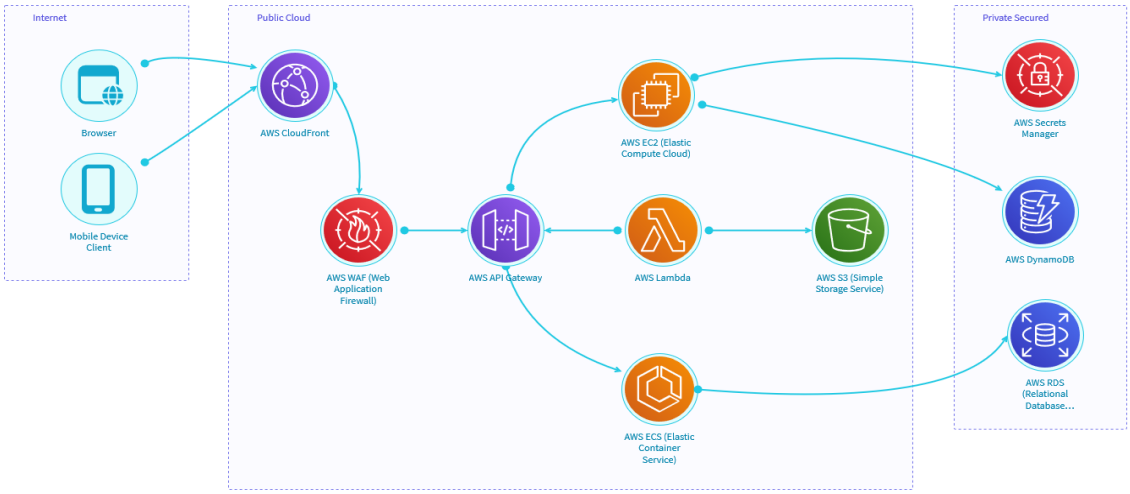


Figure 2: Topology for the proposed use case

Lambda function and a S3 bucket that support the generation of daily reports. iv) a Secret Manager that stores credentials and API keys used to connect internal requesters, such as the EC2 instance, ECS container, lambda function and databases.

4.1 Recognizing Functional Entities

To identify attack goals, a NLP model developed by Budi at al [6] that recognizes entities in a user story was employed, thus it is possible to identify subjects, actions and Personally Identifiable Information (PII) from the previously defined user stories. Listing 2 shows NER model outputs. In this way the attack tree generated later may be enhanced by enriching the attack goals with business components contributed by this NLP model.

```
1. As a logistic officer I would like to report information
about consumed, leftover and forecasted resources.
2. As a supplier manager, I would like to get information
about delivered resources to different locations.
3. As a commanding officer, I want to be able to generate
reports to provied visibility about supply chain.
```

Listing 2: Entities identified in User Stories

4.2 Modeling Threats

The previous topology feeds the Threat Modeler, in our case we used IriusRisk [1] to support the identification of threats for each component of the topology. Appendix shows the threats that were identified for each component. IriusRisk reports a large number of threats for all components of the topology, however, threats have been filtered out for EC2, ECS, Lambda and S3 components. As we can see, the Lambda function seems to be the most susceptible to threats, while the most common threat on EC2 and ECS instances is related to unauthorized access to the instance or its data. Finally, threats on S3 involve denial of service, ransomware and data exfiltration.

4.3 Composing an Attack Tree

Based on the previously defined set of threats reported by IriusRisk, a summarized attack tree that represents the different branches that an adversary can follow to achieve attack goals was generated and is represented in Figure 3.

Branch 1 represents an attack where it assumes that a Lambda function is deployed securely. However, an attacker can enumerate the endpoints of these functions and, if they are found to be vulnerable, the attacker can send malicious payloads to attempt a Server-Side Request Forgery (SSRF) attack. At this point, two paths can be taken, depending on the attack goal. The first path (left) would allow the attacker to achieve lateral movement by detecting credentials in metadata and used it to ex-filtrate reports about supply chain. The second path (right) involves the attacker gaining access to AWS S3 supply chain reports and encrypting them upon establishing a connection and generating a symmetric key.

On the other hand, Branch 2 is associated with configurations made on policies. In the event that these policies are not properly configured, an attacker can enumerate the privileges of a user and use them to perform various actions such as initiating unusual tasks on an EC2 instance or stealing credentials from profiles. In either of these cases, the attacker's ultimate goal is to escalate privileges to gain unauthorized access to the information reported by logistic officers. However, in either scenario, a countermeasure can be implemented by ensuring proper monitoring using GuardDuty. This

[1] https://www.iriusrisk.com/

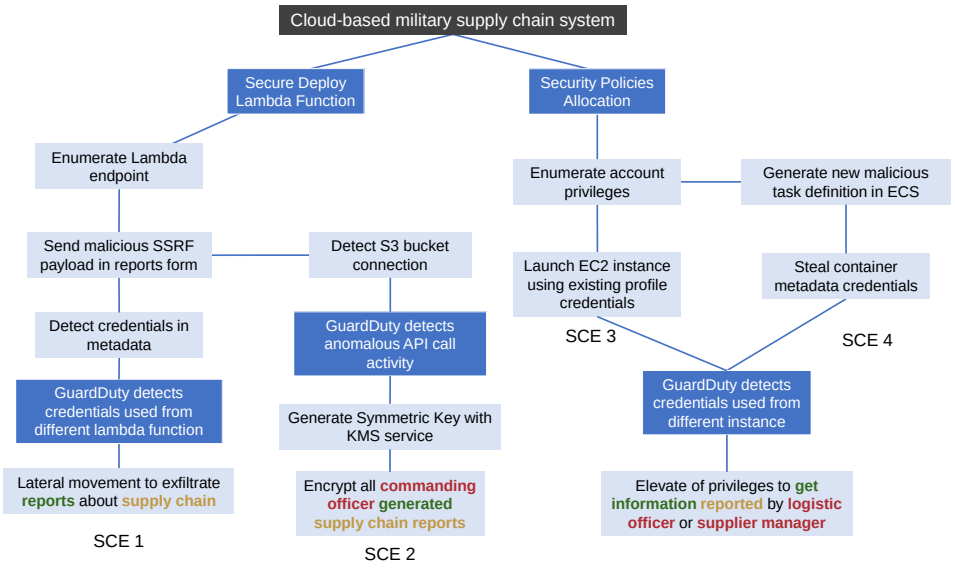


Figure 3: Attack tree

service enables the detection of credentials used in unauthorized services and also identifies unusual API calls.

For the proposed use case, we will focus on SCE expriment #3, which aims to identify whether GuardDuty is able to detect that the credentials of an EC2 instance have been used to gain access to another service and thus launch an automatic task to remove the assigned privileges [21].

4.4 Running SCE Experiment #3

The execution of the SCE experiment #3 is done according to a CE methodology: i) Definition of the behavior of the system (observability), ii) Identification of the steady state, iii) Definition of a hypothesis, iv) Execution of the experiment to validate the hypothesis. Steps 1, 2 and 3 of the methodology were already addressed in the first 4 column of Table 1. The execution of the SCE experiment (step 4) is orchestrated and controlled by ChaosXploit through an automated execution as is shown in Figure 4.

The execution of the experiment, i.e. the performing of the attack and the observation of the system, is basically composed of 3 actions: i) instance creation with a specific role, 2) ex-filtration of credentials from the instance, and finally 3) elevation of privileges to execute IAM actions such as as listing users in a IAM account. The description of each step is detailed in the following subsections.

4.4.1 *Action 1: Create instance with a specific role.*

The execution of this SCE experiment begins with the creation of an instance by a user who has permissions for execution (ec2: RunInstances) and description (ec2:DescribeInstances) of EC2 instances, as well permission to assign a IAM role (iam:PassRole). Listing 3 shows arguments used to create the instance, where particularly one highlighted in red ($iam - instance - profile$) serves to request

```
[2023-06-07 21:21:02 INFO] Rollbacks strategy: default
[2023-06-07 21:21:02 INFO] Steady state hypothesis: AWS GuardFuty detects
 the use of an instance profile credentials
[2023-06-07 21:21:02 INFO] Probe: GuarDuty-Findings
Is Steady State validated?: True  (1) Steady State Validation
Steady State validated
[2023-06-07 21:21:02 INFO] Steady state hypothesis is met!
[2023-06-07 21:21:02 INFO] Playing your experiment's method now...
[2023-06-07 21:21:02 INFO] Action: *******Creating instance*******
                          (2) Action 1: Create instance

[2023-06-07 21:21:17 INFO] Action: *******Extracting Credentials*******
-----Connecting to aws instance-----
-----Getting token-----  (3) Action 2: Connect to instance and get credentials
SUCCESS!
-----Storing credentials-----
[2023-06-07 21:21:20 INFO] Action: *******Using credentials*******
-----Reading new credentials-----
SUCCESS!
Users  (4) Action 3: Use credentials to execute IAM commands
- Arn: arn:aws:iam::173127515388:user/Sebastian
  CreateDate: 2023-05-25 20:03:33+00:00
  PasswordLastUsed: 2023-06-07 21:52:43+00:00
  Path: /
  UserID: AIDASQTZUHT6JTIVDVL3L
  UserName: Sebastian
```

Figure 4: ChaosXploit running SCE experiment #3

a specific role for this new instance. Such role (iam : $FullAccess$) allows to execute any action on IAM service.

```
1 #Instance creation
2 aws ec2 run-instances --image-id ami-02f3f602d23f1659d \
3 --instance-type t2.micro\
4 -iam-instance-profile Name=iamfullaccess --key-name demo-ec2 \
5 --security-group-ids sg-03d1a24ec4f2f11fe
```

Listing 3: Instance creation

4.4.2 *Action 2: Ex-filtration of credentials.*

Once the machine is created, a connection is established using the generic user *ec2-user* as shown at line 2 of Listing 4. Through the instance is possible to execute any action allowed by the allocated permissions. Now, instance credentials may be ex-filtrated, with the

SCE Experiment	Hypothesis	What to measure in the experiment	Scenario to be tested	Attack to execute
1	When AWS GuardDuty detects that lambda metadata credentials are used to gain access to another AWS services is possible to launch a lambda function to stop compromised service.	Events of services access using credentials from a lambda function. Detention of the lambda function after the alarm is launched.	A lambda function is implemented without sanitizing inputs it allows an attacker to get metadata credentials and gain access to another AWS services.	Lateral movement to another AWS services through a SSRF attack over a lambda function [4].
2	When AWS GuardDuty detects massive replacement operations over a S3 bucket, it preventively deny the access to the bucket.	High volume of replacement operations detected in a S3 service. Bucket access restriction after the anomalous activity is evidenced.	A lambda function is implemented without sanitizing inputs it allows an attacker to identify S3 bucket connection and kidnap supply chain reports stored in it.	Encrypt S3 (data kidnapping) through a SSRF attack over a lambda function [1].
3	When AWS GuardDuty detects that instance profile credentials are used to gain access to another AWS services, is possible to launch a lambda function to revoke misconfigured policies.	Events of services access using credentials from a EC2 instance. Detention of the instance after the alarm is launched.	A set of policies misconfigured that allows an attacker to launch a new EC2 instance passing an existing EC2 profile. If the attacker gain access to the new instance, he will get the privileges from existing EC2 profile.	Privilege escalation through launching an EC2 instance using an existing EC2 profile [21].
4	When AWS GuardDuty detects that metadata credentials are used from different ECS service, is possible to launch a lambda function to revoke misconfigured policies.	Events of services access using credentials from an ECS. Detention of the container after the alarm is launched.	A set of policies misconfigured that allows an attacker to create a new task definition with a malicious payload to retrieve ECS metadata credentials.	Privilege elevation through stealing metadata credentials in ECS instance [8].

Table 1: SCE experiments

intention of using them to obtain the IAM permission profile and create an user with the same permissions. Listing 4 shows actions performed to extract the credentials. First, a request is made to the AWS API to retrieve the authorization token (lines 5 and 6), and then a request is made to extract the credentials (lines 9 and 10). This last request returns a JSON that contains the authentication information (lines 14 to 20).

```
1  #Connection
2  ssh -i "demo-ec2.pem" ec2-user@ec2-44-202-60-62.compute
       -1.amazonaws.com
3
4  #Creating token for extraction
5  TOKEN=`curl --request PUT "http://169.254.169.254/latest/
       api/token" \
6  --header "X-aws-ec2-metadata-token-ttl-seconds: 21600"`
7
8  #Credential extraction
9  curl --write-out "\n" --request GET "http
       ://169.254.169.254/latest/meta-data/iam/security-
       credentials/iamfullaccess" \
10 --header "X-aws-ec2-metadata-token: $TOKEN"
11
12 #Output example
13 {
14   "Code" : "Success",
15   "LastUpdated" : "2012-04-26T16:39:16Z",
16   "Type" : "AWS-HMAC",
17   "AccessKeyId" : "ASIAIOSFODNN7EXAMPLE",
18   "SecretAccessKey" : "xxx/xxx/xxx",
19   "Token" : "token",
20   "Expiration" : "2017-05-17T15:09:54Z"
21 }
```

Listing 4: Exfiltration of token

4.4.3 Action 3: Elevation of privileges over IAM service. Finally, with the exposed credentials, the attacker is able to configure a new user in the AWS CLI (using `aws configure`) and with it, execute any IAM action. Figure 5 shows the execution of the command `aws iam list_users` and its respective output.

```
[sarapalacios@192 ~]$ aws iam list-users --output yaml
Users:
- Arn: arn:aws:iam::173127515388:user/Admin
  CreateDate: '2022-03-29T17:34:59+00:00'
  PasswordLastUsed: '2022-07-24T20:54:52+00:00'
  Path: /
  UserId: AIDASQTZUHT6EMUTVWHLY
  UserName: Admin
- Arn: arn:aws:iam::173127515388:user/Sebastian
  CreateDate: '2023-05-25T20:03:33+00:00'
  PasswordLastUsed: '2023-06-07T21:52:43+00:00'
  Path: /
  UserId: AIDASQTZUHT6JTIVDVL3L
  UserName: Sebastian
- Arn: arn:aws:iam::173127515388:user/vulnerable
  CreateDate: '2023-05-25T20:15:25+00:00'
```

Figure 5: Listing AWS users using the compromised credentials

The execution of the SCE experiment #3 allows to validate (or not) the hypothesis posted in Table 1. Particularly, such hypothesis is composed of a detective and corrective control. As detective control it indicates that after the occurrence of the attack, AWS GuardDuty would be able to detect that the instance profile credentials are being used to gain access to another AWS services. As corrective control it indicates that a Lambda function is able to contain the ex-filtration. The acceptance or rejection of the hypothesis depends on the validation of the detective and corrective control, and the existence of such controls depends on the security maturity level of the organization that owns the system. So in this

case, executing this SCE experiment #3 in organizations, we may find system implementations that include only a detective control, only a corrective control, a detective and corrective control, or non control at all.

An example of a detective control implemented through GuardDuty is seen in Figure 6. GuardDuty is a security monitoring service provided by AWS which is able to inform about the status of an AWS environment through security findings. In the context of SCE experiment #3, if GuardDuty is properly enabled and configured, it may detect that some instance's credentials are being used from outside of the instance that originally owns them, which is an anomalous behavior. The security finding generated by GuardDuty shows resources used, origin IP, compromised access keys and invoked actions.

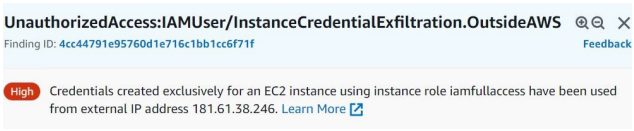


Figure 6: Screenshot of the attack detection using GuardDuty

5 CONCLUSIONS AND FUTURE WORK

New strategies to detect systems flaws are required nowadays to improve the systems security from the initial phases of the development [22]. In this way, such strategies may help to define defense mechanisms in an early way and also identify secure configurations that those mechanism should implement. In the case of a cloud environment, those security mechanisms are effectively implemented when are available by the cloud provider but also when the customer configures those mechanisms in the proper way.

This paper described a proposal to integrate SCE and NLP in the software development life cycle to contribute to build more secure systems, incorporating well-configured security mechanisms. Such a proposal is composed of a Functional Entities Recognizer, a Threat Modeler, an Attack Tree Compositor and a SCE Experiment Runner. The application of this proposal to a use case of a system deployed in the cloud and for a military-related organization showed the feasibility of our solution.

As future works we plan to extend the current attack tree with new and wider branches that include more nodes, and represent more realistic scenarios with a recursive attacker. We would like to explore also how to automate the composition of the attack from the inputs coming from the Functional Entities Recognizer and the Threat Modeler.

ACKNOWLEDGMENTS

This work has been supported by Universidad del Rosario (Bogotá) through a "Beca de Estancia de Docencia e Investigación - EDI 2022-1", and by the Spanish Ministry of Universities linked to the European Union through the NextGenerationEU program, under Margarita Salas postdoctoral fellowship (172/MSJD/22).

REFERENCES

[1] AWS. 2023. The anatomy of ransomware event targeting data residing in Amazon S3. https://aws.amazon.com/es/blogs/security/anatomy-of-a-ransomware-event-targeting-data-in-amazon-s3/. Last time accessed: 2023-06-31.
[2] Thomas Bailey, Patrick Marchione, Peter Swartz, Raed Salih, Michael Clark, and Robert Denz. 2022. Measuring resiliency of system of systems using chaos engineering experiments. In *2022 SPIE 12117, Disruptive Technologies in Information Sciences VI*, Vol. 1211704. 26. https://doi.org/10.1117/12.2632779
[3] Ali Basiri, Niosha Behnam, Ruud de Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds, and Casey Rosenthal. 2016. Chaos Engineering. *IEEE Software* 33, 3 (2016), 35–41. https://doi.org/10.1109/MS.2016.60
[4] Hacking The Cloud. 2023. Steal IAM Credentials and Event Data from Lambda. https://hackingthe.cloud/aws/exploitation/lambda-steal-iam-credentials/. Last time accessed: 2023-06-31.
[5] Tiberiu-Marian Georgescu. 2020. Natural Language Processing Model for Automatic Analysis of Cybersecurity-Related Documents. *Symmetry* 12, 3 (2020). https://doi.org/10.3390/sym12030354
[6] Guntur Budi Herwanto, Gerald Quirchmayr, and A Min Tjoa. 2021. A Named Entity Recognition Based Approach for Privacy Requirements Engineering. In *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*. 406–411. https://doi.org/10.1109/REW53955.2021.00072
[7] Ugur Koc, Parsa Saadatpanah, Jeffrey S. Foster, and Adam A. Porter. 2017. 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* (Barcelona, Spain) *(MAPL 2017)*. Association for Computing Machinery, New York, NY, USA, 35–42. https://doi.org/10.1145/3088525.3088675
[8] Rhino Security Labs. 2023. Weaponizing AWS ECS Task Definitions to Steal Credentials From Running Containers. https://rhinosecuritylabs.com/aws/weaponizing-ecs-task-definitions-steal-credentials-running-containers/. Last time accessed: 2023-06-31.
[9] Phu X. Mai, Fabrizio Pastore, Arda Goknil, and Lionel C. Briand. 2018. A Natural Language Programming Approach for Requirements-Based Security Testing. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*. 58–69. https://doi.org/10.1109/ISSRE.2018.00017
[10] Optum. 2019. ChaoSlingr: Introducing Security into Chaos Testing. https://github.com/Optum/ChaoSlingr. Last time accessed: 2022-11-9.
[11] S. Palacios, D. Díaz-López, and P. Nespoli. 2022. ChaosXploit: A Security Chaos Engineering framework based on Attack Trees. In *VII Jornadas Nacionales de Investigación en Ciberseguridad (JNIC)*, Vol. 01. Bilbao, Spain, 130–137. https://2022.jnic.es/Actas_JNIC_2022_v11.pdf
[12] Sara Palacios, Pantaleone Nespoli, Daniel Díaz-López, and Yury Niño Roa. 2023. 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). https://doi.org/10.3390/bdcc7010001
[13] Julián Ramírez Sánchez, Alejandra Campo-Archbold, Andrés Zapata Rozo, Daniel Díaz-López, Javier Pastor-Galindo, Félix Gómez Mármol, and Julián Aponte Díaz. 2021. Uncovering cybercrimes in social media through natural language processing. *Complexity* 2021 (2021), 1–15.
[14] Aaron Rinehart and Kelly Shortridge. 2021. *Security Chaos Engineering Gaining Confidence in Resilience and Safety at Speed and Scale.* Technical Report.
[15] Salah Sharieh and Alexander Ferworn. 2021. Securing APIs and Chaos Engineering. In *2021 IEEE Conference on Communications and Network Security (CNS)*. 290–294. https://doi.org/10.1109/CNS53000.2021.9705049
[16] Kelly Shortridge. 2022. From Lemons to Peaches: Improving Security ROI through Security Chaos Engineering. In *2022 IEEE Secure Development Conference (SecDev)*. 59–60. https://doi.org/10.1109/SecDev53368.2022.00021
[17] Kanchan Singh, Sakshi S Grover, and Ranjini Kishen Kumar. 2022. Cyber Security Vulnerability Detection Using Natural Language Processing. In *2022 IEEE World AI IoT Congress (AIIoT)*. 174–178. https://doi.org/10.1109/AIIoT54504.2022.9817336
[18] Julián Ramírez Sánchez, Alejandra Campo-Archbold, Andrés Zapata Rozo, Daniel Díaz-López, Javier Pastor-Galindo, Félix Gómez Mármol, and Julián Aponte Díaz. 2022. On the Power of Social Networks to Analyze Threatening Trends. *IEEE Internet Computing* 26, 2 (2022), 19–26. https://doi.org/10.1109/MIC.2022.3154712
[19] K. A. Torkura, Muhammad Sukmana, Feng Cheng, and Christoph Meinel. 2021. Continuous auditing and threat detection in multi-cloud infrastructure. *Computers and Security* 102 (2021), 102124. https://doi.org/10.1016/j.cose.2020.102124
[20] Kennedy A. Torkura, Muhammad I.H. Sukmana, Feng Cheng, and Christoph Meinel. 2020. CloudStrike: Chaos Engineering for Security and Resiliency in Cloud Infrastructure. *IEEE Access* 8 (2020), 123044–123060. https://doi.org/10.1109/ACCESS.2020.3007338
[21] Hack Tricks. 2023. The anatomy of ransomware event targeting data residing in Amazon S3. https://cloud.hacktricks.xyz/pentesting-cloud/aws-security/aws-privilege-escalation/aws-ec2-privesc#iam-passrole-ec2-runinstances. Last time accessed: 2023-06-31.
[22] David Useche, Maria Bueno, and Daniel Díaz-López. 2017. The Owasp Enterprise Security Api (ESAPI): Security Control Library. In *Owasp Latam Tour 2017.*

Component	Threat APPENDIX	Reference
AWS EC2	Attackers can abuse functionality on the server to read or update internal resources	CWE-749
AWS EC2	Attackers gain unauthorized access to data on EC2 instances	CWE-200
AWS EC2	Attackers gain unauthorized connection to the resources	CWE-284
AWS EC2	Attackers perform a denial of service	CWE-400
AWS EC2	Exploitation of insufficient logging and monitoring	CWE-778
AWS EC2	Sensitive data is compromised through unauthorized access to data	CWE-200
AWS ECS	Attackers gain unauthorized access to the control of the environment	CWE-284
AWS ECS	Attackers gain unauthorized access to data on ECS instances	CWE-200
AWS ECS	Exploitation of insufficient logging and monitoring	CWE-778
AWS ECS	Attackers gain unauthorized network access	CWE-284
AWS Lambda	Attackers gain access to a Lambda function through a Cross-site scripting attack on the DynamoDB API	CWE-79
AWS Lambda	Attackers gain access to the AWS account through leaked credentials	CWE-522
AWS Lambda	Authentication Bypass	CWE-288
AWS Lambda	Privilege Abuse	CWE-250
AWS Lambda	Privilege Escalation	CWE-269
AWS Lambda	Attackers force to increase the Amazon billing through overloading the resources	CWE-770
AWS Lambda	Attackers subvert the intended workflow of the application in order to perform unauthorized operations	CWE-840
AWS Lambda	Excessive Allocation	CWE-770
AWS Lambda	Lambda function is compromised through a code injection attack	CWE-94
AWS Lambda	Lambda function is exploited by a non-HTTP event through code injection	CWE-94
AWS Lambda	Lambda functions are accessible through an unclean container	CWE-404
AWS Lambda	Man in the Middle Attack	CWE-300
AWS Lambda	An attacker injects, manipulates or forges malicious log entries in the log file	CWE-117
AWS Lambda	Attackers exploit an unused function to enter the system	CWE-477
AWS Lambda	Sensitive information is leaked from data stores	CWE-200
AWS Lambda	Sensitive data is leaked through non-sanitized debugging/logging statements	CWE-1295
AWS S3	Attackers perform a denial of service	CWE-400
AWS S3	Data is intentionally encrypted or accidentally deleted	CWE-693
AWS S3	Sensitive data is compromised through unauthorized access to data	CWE-200

Table 2: Threats identified for component

0.4.3 Paper presented at Computer & Security journal

Computers & Security is one of the most respected computer security magazines, recognized worldwide as the premier reference source for computer security research and application expertise. This paper was submitted on December 15, 2023, to the journal and is currently under review. More information about Computers & Security may be found at `https://www.sciencedirect.com/journal/computers-and-security`

This paper proposes a new mechanism to generate the threat model based on attack-defense trees using Large Language Models (LLMs), thus automating the initial phase of the activities proposed in BSIDES and ARES. In addition, SCE experiments applicable to a DevSecOps practice are defined, which allows demonstration of the great benefit of implementing SCE on automated and modern software construction environments.

Enhancing DevSecOps practice with Large Language Model and Security Chaos Engineering

Martin Bedoya[a], Sara Palacios[a], Daniel Díaz-López[a,c], Estefania Laverde[a], Pantaleone Nespoli[b,*]

[a]*School of Engineering, Science and Technology, Universidad del Rosario, Bogota, 111711, Cundinamarca, Colombia*
[b]*Department of Information and Communications Engineering, University of Murcia, Murcia, 30100, Murcia, Spain*
[c]*Tandon School of Engineering, New York University, Brooklyn, 11201, New York, USA*

Abstract

Recently, the DevSecOps practice has improved companies' ability to produce secure software agilely, decreasing problems and improving their return on investment. Nevertheless, over-reliance on security tools can allow vulnerabilities to be implemented at different software lifecycle stages. Thus, this paper proposes the integration of artificial intelligence to automate threat discovery at the design stage and Security Chaos Engineering to support the identification of security flaws that may be undetected by security tools. A particular use case is described to demonstrate how the proposal can be applied to a retail company that needs to rapidly produce secure software.

Keywords: DevSecOps, Language Large Models, Attack-Defense Trees, Security Chaos Engineering, Cloud Computing, Software Security
PACS: 0000, 1111
2000 MSC: 0000, 1111

1. Introduction

In the current landscape, cybersecurity has emerged as a paramount concern, given the escalating frequency and magnitude of modern cyberattacks. That is, the proliferation of cybercrime has positioned it as a highly lucrative enterprise within the global black market [1]. Faced with this alarming reality, both public and private entities dedicate substantial resources to fortify their infrastructures and products, striving to counteract potential disruptive threats [2]. In this context, it is crucial to guarantee the security of software development across its entire life cycle, i.e., from the requirements to the deployment and maintenance, avoiding critical consequences in later stages. Probably, the most remarking example in this sense is the SolarWinds attack in late 2020, where malicious actors inserted a vulnerability (i.e., the Sunburst malware) into the software updates for SolarWinds' Orion platform, allowing them to conduct a widespread and highly sophisticated supply chain attack. The impact was tremendous, affecting numerous organizations and government agencies that used SolarWinds' software [3].

Besides, developing systems based on user stories is considered a significant breakthrough in the software engineering process. Typically written in natural language, these stories describe the actions that can be executed by a user in an application. Security in the software design phase is essential to ensure the early identification of security threats that may affect the system [4]. Nevertheless, it is clear that the securitization of the software development life cycle (SDLC) poses several challenges. Primarily, challenges arise from inherent subjectivity and the insufficiency of automated mechanisms for early threat identification during the SDLC. The subjective nature is particularly prominent, reflecting the dependence on the analyst's cybersecurity expertise and understanding of the business context. Moreover, the prevalent time constraints, often imposed by agile practices and strict Time-To-Market (TTM), add a layer of complexity, demanding security considerations to harmonize seamlessly with the accelerated rhythm of development [5]. In this context, one could say that automation in detecting both code and infrastructure vulnerabilities stands as a key challenge, highlighting the necessity for sophisticated tools capable of comprehensively identifying and addressing potential security flaws [6]. Furthermore, the diverse landscape of SDLC methodologies introduces a layer of confusion, requiring experienced users to tailor security measures effectively.

In a Software Security Development Life Cycle (SSDLC), an organization defines a series of security activities to be executed in the different phases of the software process. In the design phase, the construction of deliverables is necessary, which serves as input for architects and programmers to implement security controls during the development and deployment phases. These deliverables apply to traditional software development methodologies, such as the waterfall or spiral, as well as agile methodologies. One of the most common artifacts in an SSDLC is threat models, which aim to identify and map threats that may affect the different modules and components of the software. The construction of a threat model involves a certain degree of subjectivity, depending on the experience of the cybersecurity analyst or their understanding of the business. This

*Corresponding author

subjectivity leads to a certain degree of bias, making it possible for the analyst to ignore some threats, considering them improbable or simply due to lack of knowledge.

Recently, a novel paradigm named Security Chaos Engineering (SCE) has arisen in secure software and system development based on Chaos Engineering (CE) [7]. In this sense, the application of Security Chaos Engineering SCE within the DevSecOps paradigm can be seen as an effective and complementary strategy for integrating security practices into software development. In fact, if DevSecOps aims to fortify the resilience of software systems against evolving cyber threats [8], the inherent complexity of modern software architectures, together with the dynamic nature of such threats, necessitates innovative and proactive security measures. SCE proposes a distinctive methodology that aligns with the objectives of DevSecOps by proactively injecting controlled and measured security experiments into the software ecosystem [9]. This approach enables organizations to detect vulnerabilities, assess the robustness of security controls, and strengthen the overall cyber resilience of their applications.

SCE methodology has been successfully applied to various contexts (e.g., cloud environment [10], demonstrating its capabilities and applicability, even if automatizing its operations still poses a severe problem. In particular, the manual crafting of attack trees, a fundamental component of SCE, has remained a notable impediment to achieving a fully automated workflow. Here, the integration of attacks based on Language Model (LLMs) into the SCE ecosystem offers a promising path for addressing such a challenge. Leveraging the recent advancements in LLM, particularly those characterized by their natural language understanding and generation capacities, it can be possible to empower the automated creation of elaborate attack trees.

Bearing in mind those mentioned above, this article proposes an SCE-powered framework to secure software development based on user stories as an integral component of the DevSecOps paradigm, as extension of the work presented in [10]. Starting from a set of user stories of a certain application, the proposed framework applies LLM to create fine-granularity attack trees, which in turn serve as input to the SCE component. The latter is applied to the real system to test whether the proposed security controls are effective against the identified threats. The integration of these two fields (i.e., LLM and SCE) into the DevSecOps paradigm has the potential to effectively enhance security testing by incorporating proactive experimentation and the identification of security control failures in an early phase [11]. By combining artificial intelligence techniques with chaos engineering principles, researchers can explore new avenues for improving the robustness and resilience of software systems against cyber threats.

The remainder of this article is as follows. Section 2 reviews the most prominent works in the field of SCE and NLP applied to security problems. In Section 3, core concepts fundamental to the proposed framework are presented. Then, Section 4 describes our proposal, detailing the different modules. Section 5 presents the experimental results of applying the framework to a cloud-based military environment. Finally, Section 6 concludes the article, presenting some interesting future research paths.

2. Related works

Some proposals have been suggested for securing SDLC leveraging AI advancement, while others have been presented to improve security testing thanks to the SCE methodology. To this extent, the most prominent ones are analyzed and compared in the following sections.

2.1. AI-powered proposals to secure SDLC

Regarding the automation of security activities during the SDLC, the industry has focused on developing static analysis (Static Application Security Testing, SAST) and dynamic analysis (Dynamic Application Security Testing, DAST) tools to identify potential application vulnerabilities. Besides, there has been a focus on intrusion detection tools such as Web Application Firewalls (WAFs). By leveraging techniques in artificial intelligence and deep learning, these tools have significantly improved their detection rates and reduced false positives [12].

In this context, researchers in [13] leveraged Machine Learning (ML) techniques for the verification and monitoring phases within the DevSecOps loop. Specifically, they introduced a tool, namely IaC Scan Runner, designed to scrutinize various state-of-the-art Infrastructure-as-Code (IaC) languages during the application design phase. Additionally, they introduced a run-time anomaly detection tool named LOMOS. The authors proceeded to outline potential application scenarios where the integrated workflow of these tools could yield promising results.

In recent years, an outstanding AI tool has surfaced in the form of Large Language Models (LLMs), which undergo extensive training on vast datasets gathered from the Internet. These models exhibit the capability to provide real-time responses to inquiries presented by human analysts, utilizing natural language processing techniques. Such a process is often referred to as Generative Artificial Intelligence (GenAI). LLMs have been successfully applied to several cybersecurity problems, demonstrating how those systems can be helpful in defending against cyber threats [14]. In particular, some works explore the applicability of LLM to the securitization of software. Nevertheless, the research surrounding this topic is quite recent, thus the number of publications is still scarce.

To this extent, a literature review and focus groups have been conducted in [15], focusing on GenAI for Software Engineering teaching. The research outcomes show that it is possible to explore the adoption of GenAI in partial automation and support decision-making in all software development activities. Unfortunately, the code life cycle's security aspect is not considered in this research work.

Likewise, scholars in [16] systematically outlined the constraints, challenges, risks, and opportunities associated with GenAI within cybersecurity and privacy ecosystems. Beyond explaining potential malicious applications of ChatGPT from

an offensive stance, the authors emphasized the defensive aspect, wherein the tool serves to enhance cybersecurity protocols, automate processes, detect attacks, and facilitate both secure code generation and detection. The study also examined the dimensions of social, legal, and ethical considerations arising from the deployment of ChatGPT in these contexts.

One notable instance involves enhancing the security of a web application through source code analysis utilizing GPT, as outlined in [17]. Precisely, the authors introduced a vulnerability detection pipeline leveraging the GPT API, showcasing the GPT-4 model's capability to achieve up to 88.76% accuracy in identifying various Common Weakness Enumerations (CWEs) within the source code. It is crucial to highlight that the study involved testing both the GPT-3.5 and GPT-4 models, with the latter demonstrating superior performance, especially in reducing false positives. The authors conclude by proposing an extension of their work to encompass new vulnerability types and a broader array of programming languages.

Similarly, authors in [18] explored strategies designers should employ to guide ChatGPT toward recommending secure hardware code generation. In particular, the conducted analysis involved prompting ChatGPT with code scenarios aligned with CWEs, especially focusing on the hardware design (CWE-1194) aspect from MITRE. The study first illustrates instances where ChatGPT generates insecure code due to the diversity of prompts. Finally, the work proposes techniques for designers to adopt in order to generate secure hardware code, providing solutions for 10 notable CWEs under the hardware design view as listed on the MITRE site.

Another application of the GPT-4 model for cybersecurity is the generation of Governance, Risk, and Compliance (GRC) policies to prevent ransomware attacks [19]. In this study, the authors examine the model's ability to adapt to a range of vulnerability scenarios in order to describe how a GRC perspective can reduce risk. To compare and evaluate the effectiveness of the policies generated by the model, they employed different mechanisms such as game theory, cost-benefit analysis, coverage ratio, and multi-objective optimization. The authors also explore the legal and ethical component in terms of compliance.

Recently, Gadyatskaya *et al.* [20] proposed a method to create Attack Trees using LLM. The main motivations expressed by the authors are that, on one side, the manual creation of those trees is a time-consuming and error-prone process, while, on the other side, automatic tree generation techniques [21] require too much information about the underlying system. After introducing the fundamentals of Attack Trees, the authors described the methodology for synthesizing them using ChatGPT. Then, the methodology was tested in two relevant scenarios gathered from the literature, i.e., stealing a painting from a museum and tampering with a power meter to reduce the electricity bill. In both cases, the LLM tool is able to generate the corresponding Attack Trees, and the limitations of the methodology are discussed.

As illustrated in Table 1, our proposition distinguishes itself thanks to the application of a revolutionary AI technique, e.g., LLMs. Specifically, we leverage LLMs to intricately construct realistic Attack Trees, fortifying software security in the early stages of development. What sets our approach apart is its inherent integration into the DevSecOps life cycle, seamlessly interfacing with a full-fledged chaotic module, e.g., the Security Chaos Engineering SCE framework. This integration ensures a continuum of security measures across various phases of the SDLC, ranging from the testing phase to operational deployment. This holistic incorporation of AI-driven Attack Trees and SCE aligns with establishing a robust and adaptive security posture throughout the software development and deployment pipeline

2.2. SCE-powered proposals

Lately, there has been a recent shift in the chaotic methodology, with SCE emphasizing testing security controls to enhance system protection. SCE employs proactive experiments to build confidence in the system's ability to defend against potential threats and vulnerabilities. This shift acknowledges the inevitability of security failures and seeks to prioritize the evaluation of security measures.

In this sense, the open-source software ChaoSlingr represents a significant advancement in applying chaotic principles to information security [22]. This tool, designed for use on the AWS platform, offers a simplified approach to designing and executing security chaos experiments [11]. Its effectiveness has led to its adoption by several companies seeking to leverage chaotic experiments within their systems [9].

Additionally, researchers in [23] employed the chaotic methodology to strengthen the resilience of Cyber-Physical Systems (CPSs). Recognizing the complex demand for CPSs to concurrently fulfill criteria such as availability, security, safety, and reliability against varied threats sustainably, the authors introduced a formal model for CE. This model elucidates the control-theoretical aspects of CE within a non-linear industrial CPS context. Afterward, the proposed model underwent empirical validation through application to a real-world use-case scenario centered around the Tennessee Eastman Challenge (TEC) industrial process control model [24].

Besides, Torkura *et al.* proposed CloudStrike, an architecture that applies Risk-Driven Fault Injection (RDFI) for evaluating cloud security [25]. RDFI extends the CE paradigm to include cloud environments and leverages attack graphs to inject security faults. The authors tested CloudStrike on AWS and Google Cloud Platform, calculating risk values using the Common Vulnerability Scoring System (CVSS). Additionally, they used SCE methodology to assess CSBAuditor, a cloud security framework that continuously monitors cloud infrastructures to detect suspicious activities [26].

Moreover, SCE's application in improving API security is discussed in [27]. Particularly, the authors propose utilizing SCE to evaluate the security controls configuration of APIs, thereby identifying vulnerabilities early. By focusing on the OWASP top 10 critical web application security risks and automated attack detection, they advocate for SCE experiments to tackle these challenges.

Also, SCE experiments were conducted to assess the robustness of System of Systems (SoS) against potential attack-

Table 1: Comparison of the AI related works highlighting the main features.

Related work	Description	AI-related technique	SW appl.	Security appl.	DevSecOps appl.
Cankar *et al.* [13]	IaC Scanner	NLP	≈	✓	✓
Nguyen-Duc *et al.* [15]	GenAI impact on SW engineering courses	LLM	✓	✗	✗
Gupta *et al.* [16]	LLM application to cybersecurity	LLM	≈	✓	✗
Szabó *et al.* [17]	Web application security	LLM	✓	✓	✗
Nair *et al.* [18]	Secure HW code	LLM	✓	✓	✗
McIntosh *et al.* [19]	GCR compliance	LLM	≈	✓	✗
Gadyatskaya *et al.* [20]	Automatic Attack Tree creation	LLM	≈	✓	✗
Our proposal	Automatic Attack Tree creation	LLM	✓	✓	✓

Legend: ✓ Yes – ✗ No – ≈ Partially

ers in [28]. Using the Chaos Toolkit, CE and SCE experiments were performed on a Virtual Unmanned Aerial Vehicle (VUAV), leveraging Attack Trees to model attacker actions. Five experiments were executed, measuring CPU and RAM usage as performance indicators.

The paper in [29] discusses the application of SCE from an economic viewpoint. Concretely, the author emphasizes that security Return On Investment (ROI) commonly relies on the successful prevention of attacks, while they are almost inevitable. In this sense, SCE culture helps in shifting the paradigm from avoiding the attacks to minimizing their impact through continuous and rigorous experimentation, system resilience, and better response plans.

It is worth remarking that, in our previous research, we introduced ChaosXploit [30], an innovative framework empowered by SCE principles. ChaosXploit leverages a comprehensive knowledge database consisting of attack trees to identify and expose vulnerabilities in a certain system or software. Utilizing ChaosXploit can form an integral component of a proactive defensive security strategy, enabling the timely detection and rectification of software misconfigurations. Then, the work has been further extended, proposing a framework combining the capabilities of SCE and NLP [10]. In particular, the framework incorporates a NLP model designed to extract pertinent information from a collection of user stories associated with a specific application. By leveraging this extracted information, the proposal models and distinguishes potential threats that could impact the software, identifying early-stage attack strategies employed by cybercriminals. Then, the SCE methodology was implemented on the actual system to evaluate the effectiveness of the proposed security controls against the recognized threats.

From the SCE standpoint, as reported in Table 2 our presented framework stands out from preceding solutions due to its remarkable versatility. It demonstrates adaptability across diverse contexts, encompassing various cloud platforms like AWS, GCP, and Azure, as well as containers such as Docker and Kubernetes, among others. Moreover, it extends its utility to web applications. An additional distinctive feature lies in its elevated level of automation. We introduce a preconfigured series of actions that can be systematically executed upon experiment initiation, streamlining the process and minimizing the need for manual intervention. This impressive applicability and automation contribute to the efficacy and broad utility of our proposed framework.

Nonetheless, one of the main drawbacks discussed in [10] was indeed the manual crafting of Attack Trees. As previously mentioned, the manual creation of those trees presents several disadvantages that decrease the scalability of the solution to modern systems or complex software. To solve such an issue, we leverage the outstanding capabilities of LLM, which is integrated into the DevSecOps cycle to secure software from the early stages. Furthermore, to the best of our knowledge, it is noteworthy that none of the contemporary proposals integrates the advanced capabilities of AI with the distinct advantages offered by SCE in the cybersecurity ecosystem. This represents a significant gap in the existing research landscape, as the synergistic fusion of AI technologies with the proven benefits of SCE holds substantial potential to enhance the effectiveness and efficiency of cybersecurity measures.

3. Background

For the sake of the reader, it is crucial to introduce some core concepts. That is, Section 3.1 describes LLM fundamentals, then Section 3.2 presents the SCE basics, and finally Section 3.3 shows the main ideas of attack modeling.

Table 2: Comparison of the SCE related works highlighting the main features.

Related work	Attributes	Appl. Context	CE Tool	Threat Model	Experiment Data	Automation Level
Konstantinou *et al.* [23]	Resilience	CPS	Ad-hoc	✗	crafted	✗
Torkura *et al.* [25]	Security	Cloud	Ad-hoc	Could Attack Graphs	crafted	≈
Sharieh, Ferwron [27]	Security	API	✗	✗	✗	✗
Bailey *et al.* [28]	Security	SoS	ChaosToolkit	Attack Trees	crafted	✗
Shortridge *et al.* [29]	Security	IT systems	✗	✗	✗	✗
Our proposal	Security	Any	ChaosToolkit based	Attack Trees	Public buckets	✓

Legend: ✓ Yes – ✗ No – ≈ Partially

3.1. Large Language Models (LLMs)

NLP is a critical branch of artificial intelligence focused on analyzing and understanding human language for specific task resolution. Combining techniques from linguistics, computer science, and machine learning, NLP empowers computers to process and interpret natural language in both textual and spoken forms, addressing tasks like syntactic and semantic analysis, named entity recognition, sentiment analysis, and language translation.

Large Language Models (LLMs) constitute an artificial intelligence model specifically crafted for extensive comprehension and generation of human-like language. A pivotal advancement in LLM development is the introduction of the transformer architecture, as outlined in the seminal paper "Attention is All You Need" by Google researchers in 2017 [31]. This architectural innovation revolutionized the field of Natural Language Processing (NLP) by enabling models to concurrently consider the entire context of a word sequence, departing from the traditional sequential processing approach. The parallelization inherent in transformers significantly enhanced the efficiency and performance of language models.

These models offer significant potential to enhance cybersecurity by enabling professionals to analyze vast amounts of textual data for threat detection and classification. Leveraging the effective processing of natural language, LLMs act as multiplier effects in cybersecurity operations, aiding teams in managing the increasing volume of security threats and data [32].

3.2. Security Chaos Engineering (SCE)

As previously mentioned, CE facilitates examining systems under the principle that valuable lessons can be learned from failures. The core focus of CE principles is to assess the resilience of the underlying system by introducing controlled chaos. Beyond this, the notion of SCE emerges, defined as "the proactive experimentation to identify security control failures and to build confidence in the system's ability to defend against malicious conditions in production" [11].

Within the cybersecurity ecosystem, the CE paradigm has experimented with limited involvement. However, ChaosXploit stands out as an innovative SCE framework by enabling the execution of chaos engineering experiments using attack trees as a defensive security strategy to detect misconfigurations in software [30]. Furthermore, additional research on ChaosXploit has expanded its experiment base, incorporating artificial intelligence techniques such as entity recognition algorithms to better identify potential threats within a system[10].

It has to be stated that, overall, the SCE experimentation methodology is similar to that of CE, incorporating steady state, observability, and hypothesis during tests. The first step involves establishing a steady state, where the system is observed in a controlled environment to understand its baseline behavior. Subsequently, observability is maintained throughout experiments, enabling continuous monitoring and analysis of the system's response to simulated security threats. This emphasis on observability is crucial for systematically assessing the impact of potential vulnerabilities introduced during experimentation. Finally, hypothesis formulation during tests guides targeted investigations into security flaws caused by human factors, insecure design, and inherent system resilience limitations [9].

Moreover, SCE serves as a proactive strategy to enhance both software and infrastructure security. In software security, SCE experiments could involve intentionally injecting vulnerabilities, like SQL injection, into targeted modules, evaluating the resilience of code, components, and incident response mechanisms. Simultaneously, in infrastructure security, SCE experiments may simulate unexpected events such as DDoS attacks, assessing the network's response to sudden surges in traffic. By intentionally introducing controlled chaos, organizations gain insights into vulnerabilities and weaknesses, enabling them to fortify their overall security posture, whether in code, components, or network infrastructure. This unified approach ensures a holistic enhancement of security resilience by identifying and addressing potential threats in both software and infrastructure domains.

3.3. Attack Modeling

Attack Representation Modeling (ARM) can be defined as crucial in the cybersecurity ecosystem. ARM systematically identifies relationships between individual security alerts generated by various security components (e.g., Intrusion Detection Systems (IDSs), firewalls, etc.). Unlike simply describing the sequential steps of an attack, ARM's primary objective is to determine the path of an attack and generate comprehensive reports. That is, this approach enables cybersecurity professionals to gain insights into the nature of a threat, facilitating effective response strategies [33]. Specifically, Cheung *et al.* [34] have outlined a structured methodology for developing a model capable of successfully recognizing cyber attack scenarios. In particular, the process begins with identifying specific attacks, further subdivided into attack subgoals until each logical facet of an attack is isolated by detection systems. Subsequently, these attacks are attributed based on observed events, system states, and interfaces, forming the initial phases of the modeling process [35].

In subsequent stages, the relationship among these identified attacks is meticulously developed. This process involves considering temporal relationships, such as the sequence in which attacks are identified, attribute-value relationships, where attacks might stem from similar sources, and prerequisite relationships, elucidating how one attack might trigger another. By incorporating these relationships into the model, the ARM captures not only individual events but also the intricate connections between them.

In order to develop a full-fledged Attack Representation Model (ARM), a crucial strategy involves utilizing attack modeling techniques like Attack Graphs and, notably, Attack Trees. The latter serves as a structured framework, portraying potential attack scenarios. Expanding on this foundation, variations such as the Attack and Protection Tree, Defense Tree, and Attack-Defense Tree contribute to enriching the modeling process [36].

In this sense, Attack Trees offer a structured framework for modeling threats against computer systems. These trees systematically represent potential attacks, utilizing a tree structure where the root node denotes the primary goal, and leaf nodes signify diverse methods of achieving that goal. Each node becomes a subgoal, with children representing various approaches to attain it. Particularly, OR nodes denote alternatives, and AND nodes signify different steps toward achieving the same goal. By assigning values to leaf nodes and considering factors such as specialized knowledge, time requirements, and associated risks, attack trees facilitate a systematic assessment of system security. This structured approach aids in designing effective countermeasures and making informed decisions regarding the cost-effectiveness of potential attacks [37]. Nonetheless, building an attack tree is a complex task that requires advanced knowledge of the threats that can affect a system and holistic knowledge of the system itself. This process can be susceptible to the subjectivity and bias of the person performing it; it is more likely that a red team professional will generate a tree with a tendency to affect the system, while a blue team professional will generate it with a defensive tendency.

Attack trees and Attack Graphs are commonly employed to model and analyze attacks in computer networks. As mentioned previously, several alternative tree structures, including Attack and Protection trees, Defense Trees, and Attack-Defense trees, have been developed to explore attacks from both offensive and defensive perspectives. For instance, Edge *et al.* [38] introduced the Protection Tree, which identifies protection areas based on impact, probability, and cost calculations derived from an attack tree. In contrast, Bistarelli *et al.* [39] extended attack trees to create Defense Trees, measuring the return of actions for both attackers and defenders using indexes like Return on Investment (ROI) and Return on Attack (ROA). Unlike traditional attack trees, both Defense Trees and Attack-Defense Trees integrate nodes for actions related to both attack and defense, allowing for a comprehensive analysis of the economic effectiveness of these actions. The Attack-Defense tree further aims to provide an overview of how offensive and defensive components converge over time.

4. SCE-LLM based proposal

The use of cloud-based technologies has revolutionized the software supply chain by streamlining the development and deployment processes. In particular, one could say that handing over the physical infrastructure management to a third party offers several widely known economic and technical obsolescence advantages [40]. By knowing the components of a software supply chain, it is possible to identify the threats that may affect them and anticipate the implementation of vulnerabilities by applying security controls. Considering the previous, we propose integrating two novel concepts, i.e., SCE and LLMs, to secure the software life cycle.

This section describes our proposal that aims to protect cloud-based systems using SCE and Artificial Intelligence when software factories use DevSecOps practices. Figure 1 depicts the main modules of the proposed framework that will be described next. Throughout this section, we will refer to the Security Analyst as a role that is assigned to a development team and performs security tasks at each phase of the software life cycle to ensure a vulnerability-free transition to production. Some of the activities of the Security Analyst include gathering security requirements, creating the threat model, analyzing the results of security tools integrated into the CI/CD pipeline, supporting problem remediation, verifying compliance with security standards, e.g., OWASP Application Security Verification Standard (ASVS) [41], among others.

Throughout the next sections, the main components of the loop illustrated will be described.

4.1. Stage 1: Requirement elicitation

DevSecOps practices have proven to be a leverage in software development processes. Automating build and deployment activities has improved time to market and ensured that all stakeholders perform their roles uniquely. When a software factory implements DevSecOps methodologies, security by design is critical. From a set of user stories and a topology of

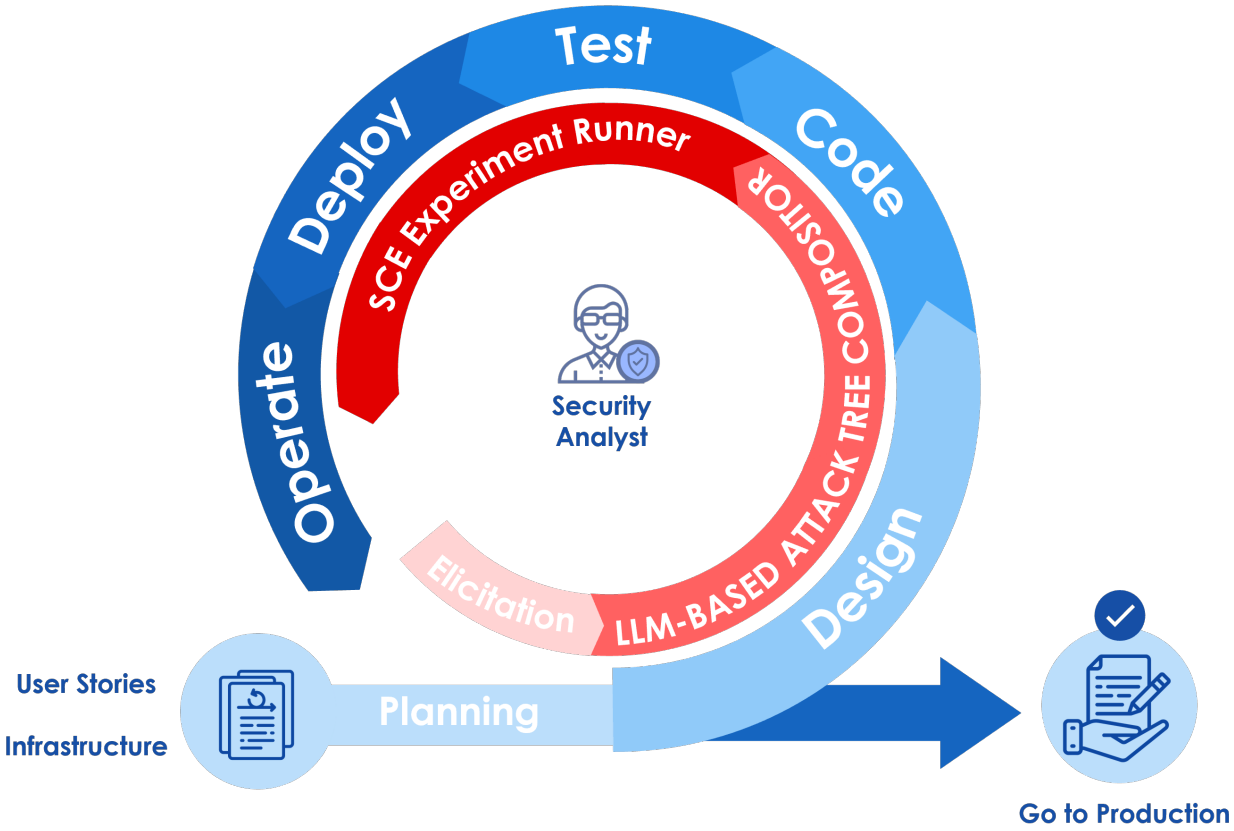


Figure 1: Integration of our proposal in a DevSecOps pipeline

the infrastructure components, a Security Analyst can identify achievable, measurable, and agile security activities that guarantee the security of the entire software life cycle [42].

In particular, user stories are essential to design a system from a functional perspective, as these allow us to comprehend the diversity of users that will interact with the system, the type of actions that the system should support, and also the type of data that the system will process [43]. Without even starting to implement a system, it is possible to estimate many activities of a new project just by reviewing in detail the user stories.

On the other hand, from a general list of software components, a Security Analyst can identify the threats and vulnerabilities that may affect each component, giving a general picture of the tactics, techniques, and procedures that an adversary can use to compromise the system. In this sense, it is easy to find databases of known vulnerabilities on specific software components commonly used in the development of cloud-based systems, which provide a quick overview of the patches and configurations to be applied in each development sprint.

By understanding the threats and vulnerabilities that can affect a system, a Security Analyst can identify the attack goals that an adversary may achieve. An attack goal is the result of a sequence of malicious actions that allow the adversary to compromise the system and are essential for Attack Tree-based threat models [44]. Attack goals are usually based on the understanding of the target system and the identification of malicious actions that can cause an operation, reputation, or legal impact, e.g., an attack goal could be the ex-filtration of information from a database.

4.2. Stage 2: LLM-based Attack-Defense Tree Compositor

Our proposal aims to demonstrate that it is possible to automate the construction of an Attack Tree-based threat model using artificial intelligence. The use of large language models for the generation of Attack Trees can be a complicated task to solve. Thus, Figure 2 shows the sequence of steps that are necessary to generate a possible Attack Tree using the LLMs. Multiple tests were performed using the GPT-3.5 and GPT-4 versions of LLMs released by OpenAI through their chat identifying overwhelming differences between them. Each step of the final diagram flow depicted in Figure 2 is described next.

4.2.1. Application Security Context

In the first instance, it is necessary to contextualize the LLM model with the concepts of application security, threat models, and Attack Trees. This task is important because it will ensure that the model understands the context of future tasks to be solved and does not generate content from other types of topics. It is possible to ask questions to the LLM model related to these topics and request a summary of the information. In this way, concise information can be concentrated in its memory, considering this is limited. Questions such as “What is application security?”, “What is threat modeling?”, and “How to

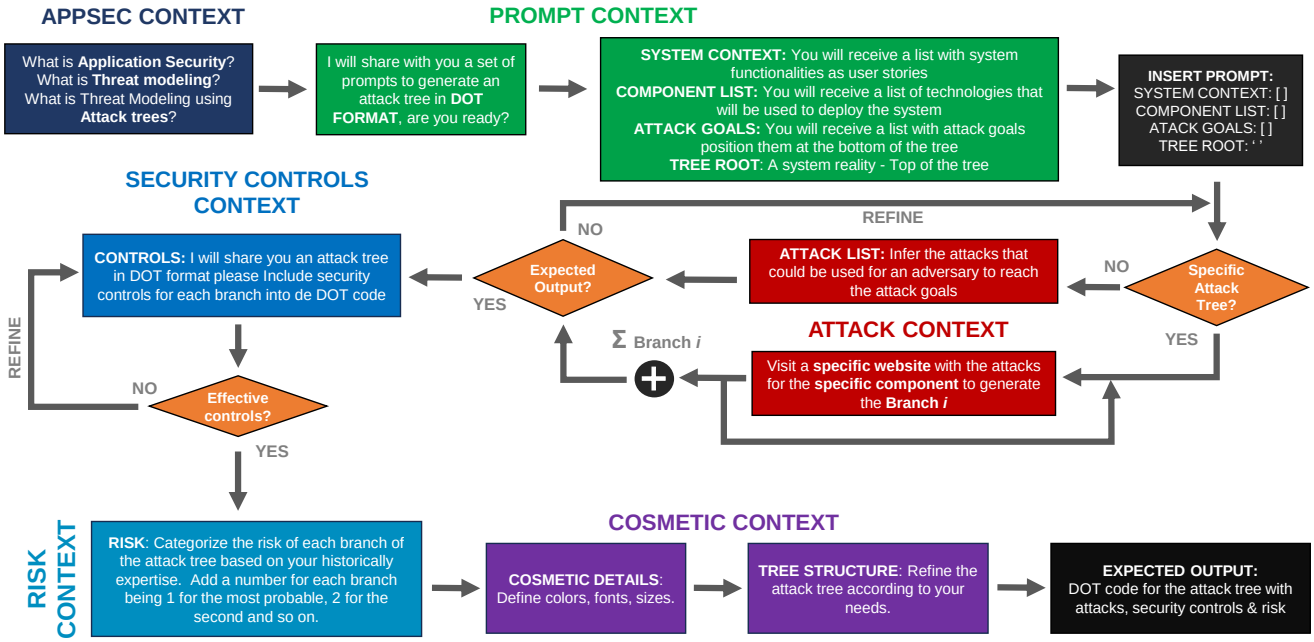


Figure 2: Proposed flow diagram to generate an Attack-Defense Tree using an LLM

perform threat modeling using an Attack Tree?" may be good approaches.

4.2.2. *Prompt Context*

Subsequently, the LLM model will receive a series of structured prompts and generate an expected output as an Attack Tree in DOT format. The defined structure of the prompt chosen for this proposal is the next one:

- **System Context**: It contains a phrase that indicates to the LLM that the context of the system will be provided in a specific form, e.g., "You will receive a list of system functionalities as user stories".
- **Component List**: it contains a phrase that indicates to the LLM that a list of infrastructure components will be provided, e.g., "You will receive a list with the technologies that will be used to deploy the system".
- **Attack Goals**: it refers to a phrase that indicates to the LLM that a set of attack goals will be provided, e.g. "You will receive a list with attack goals, position them at the bottom of the tree".
- **Tree root**: it refers to a phrase that indicates the main node to the LLM, e.g. "A system reality - Top of the tree".

The reason for the 4-parameter prompt is the result of a series of tests where it was found that adding more parameters caused LLMs to generate meaningless code. It should be noted that the order of the parameters does not affect the result.

4.2.3. *Inserting the Prompt*

This is the most important phase of the process since all the knowledge of the system under development must be summarized. Results at this step may differ between different versions of LLMs, e.g., GPT-3.5 has more limited memory, and its inference capacity is lower than GPT-4, so many user stories or technology components make GPT-3.5 generate meaningless code. An example of prompt insertion is included in Section 5 and is specifically contained in Table 4.

4.2.4. *Attack Context*

During this research, it has been discovered that it is possible to generate generalized or specific Attack Trees. In both cases, Graphviz was used to render the Attack Tree code generated by the LLM.

To create a generalized Attack Tree, asking the LLM model to infer attacks based on the system context, topological components, and attack goals may be enough. A generalized Attack Tree provides a broader view of the attacks that the system may suffer and requires the Security Analyst to identify whether the inferred attacks are actually exploitable on the system. One way to ask the LLM to generate a generalized Attack Tree is with a prompt like "Infer the attacks that could be used for an adversary to reach the attack goals based on system context and infrastructure components".

On the other hand, a specific Attack Tree provides a clearer view of the sequence of attacks the adversary must follow to reach the attack goals. This type of Attack Tree is appropriate when the Security Analyst requires specific attack procedures. Generating this kind of Attack Tree requires providing the LLM with websites that contain the attacks, e.g., `https://cloud.hacktricks.xyz/`. To generate a specific

Attack Tree, a better result will be obtained by asking it to generate the branches independently for each component and then concatenate them. Subsequently, it will be necessary to verify that the attacks generated by the LLM actually lead to the attack goals otherwise those branches should be refined. One way to ask the LLM to generate a specific Attack Tree is with a prompt like "You will receive a series of prompts with the URLs to websites when you can consult the attacks for each component".

As part of the tests performed in this research, overwhelming differences have been identified between the Attack Trees generated by the GPT-4 and GPT-3.5 models. The main difference is that GPT-3.5 can not create specific Attack Trees because it does not have the ability to query the internet. On the other hand, with respect to generalized Attack Trees, GPT-3.5 tends to repeat attacks, which have a poor relationship to system context or user stories.

4.2.5. Security Controls Context

Once the generated Attack Tree is clear for the Security Analyst, he may proceed to ask the LLM model to identify security controls for each branch as shown in Figure 1. It is also possible to request a control for each attack, but it tends to repeat controls.

It is also possible to enrich the prompt with specific websites containing information on how to secure specific components. The advantage of doing this is that the model will be able to propose specific open-source or third-party security services as controls to contain an attack.

Subsequently, if controls are adequate based on the Security Analyst's experience, the LLM model is asked to concatenate the security controls as new Attack Tree's nodes and redraw it.

4.2.6. Risk Context

A way of using an LLM to enrich an Attack Tree is to ask the LLM to analyze the cybersecurity risk for each of the Attack Tree's branches. For this research, the LLM model was asked to infer the cybersecurity risk of each branch based on the historical knowledge of security breaches. In this regard, there is an important difference in results between GPT-3.5 and GPT-4, because the latter has been trained with a larger and more up-to-date dataset and also has the ability to query the internet. For such previous reasons, results obtained from GPT-4 are highly dependent on historical security breaches in comparison to GPT-3.5.

4.2.7. Cosmetic Context

Once the Security Analyst is comfortable with the generated Attack Tree, it is possible to ask the LLM to include cosmetic changes, for example, text font, color, size, and node colors, among others. It is also possible to ask the LLM to redefine the tree's structure, as it may be necessary if the generated Attack Tree has repeated connections between nodes or if the location of the parent nodes or attack goals does not meet the requirement of the Security Analyst.

4.2.8. Expected Output

The advantage of providing the LLM structured and individual prompts with single tasks, instead of a prompt that contains multiple tasks at once, is that the LLM has a better performance solving small tasks rather than solving a large number of tasks. This is explained by the inference capability, which is limited by the number of tokens the model can process. Another advantage of providing multiple individual prompts is related to the model's recall capability, which is relevant as it allows the model to infer attacks and controls more closely related to the context of the system or infrastructure components.

If the result of the Attack Tree is not as expected, e.g., it contains meaningless attacks, faked controls, nodes without connections, or even if the relationship between attacks and goals is null, it is possible to deliver the code back to the LLM in DOT format and request specific changes as required.

An example of an Attack Tree generated using the flow proposed in this section may be seen in Figure 4, where light-blue boxes represent offensive actions done by an attacker, and dark-blue boxes represent defensive actions, i.e., security controls.

4.3. Stage 3: SCE Experiments Runner

Many companies that adopt early DevSecOps practices have an over-reliance and over-dependence on static and dynamic security tools. However, such tools may not be infallible. When more mature practices exist, ethical hacking activities, in addition to regular DevSecOps practices, are generally included, with a remarked focus on business logic. This is because setting up security tools to test business logic can be a complicated and ambiguous task [45].

In addition to the security tools that can be integrated into a DevSecOps practice, we propose the integration of SCE as a way to challenge the security of systems in the current paper. As mentioned in Section 3, SCE is based on a well-defined methodology that includes: i) observing what to measure, ii) defining a stable state, iii) proposing a hypothesis, iv) conducting an experiment, and v) analyzing the results. The hypothesis definition is one of the most important steps in the methodology as it validates security assumptions, which are not tested by traditional testing methods as are assumed as truth.

For example, one hypothesis in a DevSecOps context that could be tested using the SCE methodology is "there is a detective and corrective mechanism that will be activated when a developer self-promotes source code in a software factory". Such a hypothesis is not generally tested by traditional security tools, as those tools test predefined rules of signatures associated with known vulnerabilities in the running application, the source code, or the supporting components. For this purpose, ChaosXploit is used in the current proposal [30], an open-source framework that allows designing chaos engineering experiments and can be integrated into a DevSecOps practice. The logic behind the SCE experiment's execution can be represented in the pseudocode at Algorithm 1.

At this point, one could ask in which ways the DevSecOps paradigm can benefit SCE. In our vision, SCE allows us to define hypotheses and experiments applicable to the different

Algorithm 1 Pseudocode of the execution of a SCE experiment e_i

Load experiment e_i
Load steady state ss_i
Load hypothesis h_i
Load attack procedure $ap_i = [s_0, ... s_n]$
Load attack goal ag_i
Load rollback procedure rp_i
Validate e_i, ss_i, h_i, ap_i, ag_i and rp_i
if Validation fails **then**
    experiment e_i is interrupted
else
    continue execution of experiment e_i
end if
Evaluate Steady State ss_i
if steady state fails **then**
    Experiment e_i is interrupted
else
    Continue execution of experiment e_i
end if
for each step s_j in attack procedure ap_i **do**
    Execute s_j
    if s_j gets a runtime exception **then**
        Experiment e_i is interrupted
        Implement rollback procedure rp_i
    else
        Continue execution of experiment e_i
    end if
end for
Summarize outcomes of attack procedure ap_i
if attack goal ag_i was achieved **then**
    hypothesis h_i refuted
else
    hypothesis h_i validated
end if
Implement rollback procedures rp_i
Conclude experiment e_i
End of experiment e_i

stages of a DevSecOps practice, i.e., designing, coding, testing, deployment, operation, and monitoring. On the other hand, it is also possible to use pipelines to launch experiments in an automated way because one of the fundamental components of SCE is precisely automation.

Once the Attack Tree has been built in Section 4.2, the Security Analyst can use it as input to define SCE experiments that may be applied over the application or infrastructure (Section 4.3). Each branch of the Attack Tree is converted into an SCE experiment by defining a testing scenario where the defensive (security controls) and offensive (attack techniques) nodes converge. As will be seen in Section 5.3, the "hypothesis" of an SCE experiment is determined from the validation of the effectiveness of the defensive nodes. Additionally, the "attack to execute" in an SCE experiment is derived from the attack goal and the attack techniques contained in the tree, and the "what to measure" in an SCE experiment is defined from the traces left by every offensive node.

4.4. Toward an enhanced DevSecOps practice

Our proposal benefits the Design stage of the software supply chain because it allows the automation of the creation of a threat model based on attack-defense trees, taking into account that building a threat model is an arduous and complex task that requires a high level of technical expertise in cybersecurity. Additionally, our proposal benefits the Code stage because it is possible to propose SCE experiments on the source code to identify vulnerabilities that are not detectable with SAST tools. Finally, our proposal aids the Test and Deploy stages because SCE experiments applicable to running applications or their components can be proposed.

Thus, LLM and SCE can be used to execute unusual security tests in a DevSecOps practice, as we will see in Section 5, where we demonstrate the exploitation of weakness in some DevSecOps security stages, which are focused on the detection of vulnerabilities. Some of these exploitations are due to an over-reliance on SAST and DAST tools. Finally, these unusual security tests may be automated and integrated as stages of a CI/CD pipeline, allowing an organization can achieve deployment time objectives in a secure way.

5. Use case: Improving AWS DevSecOps using LLMs and SCE

This section describes how the proposal presented in Section 4 is applied in a concrete scenario of a software supply chain. In this scenario, a retail company, motivated by a digital transformation strategy, has decided to implement one of its critical mission systems with AWS cloud services. Among the foreseen benefits of this implementation is the possibility of adopting a cloud-native design supported by an AWS DevSecOps environment. The system chosen to be deployed is the Supply Chain Management (SCM), which has to deal with a high turnover in the company's set of vendors, customers, and partners, as well as frequently updated commercial and sales processes, resulting in the need for rapid software development

cycles, i.e. a CI/CD-based software life cycle. Security has to be guaranteed in this pipeline, independent of time constraints imposed by agile development frameworks, as operational and private/personal data security is one of the business objectives.

Thus, this section explains how may be applied the different stages of our proposal (requirement elicitation, LLM-based Attack Tree compositor, and SCE experiment runner) in the use case described previously.

5.1. Requirements Elicitation and DevSecOps environment

As a result of the software life cycle analysis phase, three critical user stories have been selected to be deployed in a quarter following an agile development framework. These user stories are represented in Listing 1.

```
1. As a logistic officer, I would like to report
    information about consumed, leftover and
    forecasted resources to manufacture the final
     products.
2. As a supplier manager, I would like to obtain
    information on products delivered to end
    customers.
3. As an operations executive, I want to be able
    to generate reports to determine supply chain
     performance.
```

Listing 1: User Stories

From the description of the use case and from previous user stories, a topology of components that support the implementation of the SCM system emerges (Figure 3). This topology describes the system integration and deployment components. The main components necessary to implement the SCM system using AWS DevSecOps environment are: i) an EC2 instance that uses an RDS service to manage transactional SCM operations, and ii) a Lambda Function and an S3 Bucket responsible for generating and storing SCM reports, iii) an AWS CodeCommit service to version the source code, iv) an AWS CodeBuild service to create the system artifacts, and v) an AWS CodeDeploy service to launch the system. In addition, there are supporting components such as AWS CodePipeline that handles CI/CD automation processes, AWS CodeGuru that analyzes source code, AWS IAM that manages authentication and authorization, and AWS CloudWatch that monitors the entire AWS environment.

5.2. Composing an Attack Tree using a LLM

The process for generating an Attack-Defense Tree was done following the steps presented in Section 4.2, and the specific inputs toward the LLM, i.e. ChatGPT-4, are contained in Table 3. The creation of this Attack Tree allows the Security Analyst an agile way to identify the techniques and procedures that an adversary may employ to reach an attack goal over the SCM system. Figure 4 shows the Attack-Defense Tree generated based on user stories and topology of AWS components presented at Section 5.1. Full execution of steps to replicate an Attack-Defense Tree as the one shown in Figure 4 can be found at: `https://chat.openai.com/share/e2ad2678-cc6b-48fd-945d-cd8600df5b5b`

Once the tree has been generated, based on the risk criteria determined by ChatGPT-4, the Security Analyst proceeds to analyze the branches. The Attack-Defense Tree analysis may be done from the most to the least risky branch, starting from Risk 1 and ending at Risk 5.

According to ChatGPT-4, the branch with the highest risk (Risk 1) in the tree is branch #2. Branch #2 describes the sequence of attacks that an adversary should follow when identifying a set of S3 misconfigured policies. The attack sequence of this branch leads to both attack goals, which makes sense due to an adversary with full control of the S3 buckets can not only exfiltrate information but also encrypt it.

Then ChatGPT-4 assigned high risk (Risk 2) to the #1 branch in the tree, the EC2 instance. Again, a misconfigured set of IAM policies can allow an adversary gaining access to the instance to steal credentials from the machine, thereby escalating privileges over the AWS environment. In particular, this would allow them to easily reach the attack goals by interacting with the other infrastructure components.

Then ChatGPT-4 assigned medium risk (Risk 3) to the #3 branch in the tree, the Lambda Function. In this case, the exploitation of vulnerabilities in the code can enable privilege escalation in the entire AWS environment compromising not only the lambda function but the entire infrastructure. However, exploitation will depend on poor coding security practices.

Then ChatGPT-4 assigned low risk (Risk 4) to the #5 branch in the tree, the AWS CodePipeline. The compromise of this component may enable unauthorized modification of the source code, which can be used to achieve attack goals. However, it requires a series of bad role and permissions management practices.

Finally, ChatGPT-4 assigned the lowest risk (Risk 5) to the #4 branch in the tree, the AWS CodeBuild. In this case, Build scripts can be manipulated by adversaries to include unauthorized execution of commands, which can enable compromise of the entire AWS environment. Exploiting this branch is complex because, in a secured pipeline, building a new artifact requires the authorization of a peer or lead developer.

5.3. Building Security Chaos Engineering Experiments

Having as input the Attack-Defense Tree implemented in the previous section and considering the SCE guidelines described in Section 4.3, the Security Analyst can propose implementable, automatic, and measurable SCE experiments. Thus, the Security Analyst may propose different SCE experiments for this use case, described in Table 4.

SCE experiment #1 is built from branch #2 and is focused on testing the capability of AWS GuardDuty to detect suspicious operations over an S3 bucket (i.e., hypothesis) in situations where misconfigured policies allow an attacker to edit S3 buckets in an unauthorized way (i.e., scenario to be tested). The suspicious operations may be massive replacements that could probably indicate that the repository storing SCM data is being encrypted by an attacker [46]. Our assumption is that there should not be misconfigured policies, and if some appear, GuardDuty should detect and prevent it. Eventually, that con-

Step	Input to the LLM
Application Security Context	What is application Security?
	What is threat modeling?
	What is threat modeling using Attack Trees?
Prompt Context	It was explained to the model that a series of prompts structures would be shared with it and that it was expected to generate an Attack Tree in DOT format
Inserting the prompt	SYSTEM CONTEXT: ”As a logistic officer, I would like to report information about consumed, leftover and forecasted resources to manufacture the final products”, ”As a supplier manager, I would like to obtain information on products delivered to end customers”, ”As an operations executive, I want to be able to generate reports to determine supply chain performance”
	COMPONENT LIST: ”AWS EC2”, ”AWS S3”, ”AWS Lambda”, ”AWS CodeBuild”, ”AWS CodePipeline”
	ATTACK GOALS: ”Ex-filtrate supply chain information”, ”Encrypt supply chain information”
	TREE ROOT: ”Cloud-based supply chain system”
Attack context (Specific Attack Tree)	You will receive a series of prompts with the URLs to website when you can consult the attacks for each components
	Start with EC2 instance: https://cloud.hacktricks.xyz/pentesting-cloud/aws-security/aws-privilege-escalation/aws-ecs-privesc generate a path with the sequence of attacks to reach the attack goals
	Concat the attacks vertically demostrating the sequence of attacks to reach the attack goals
	Nice, now generate the S3 branch of the tree: https://cloud.hacktricks.xyz/pentesting-cloud/aws-security/aws-privilege-escalation/aws-s3-privesc/ generate a path with the sequence of attacks to reach the attack goals.
	Nice, now generate the LAMBDA branch of the tree visit the website: https://cloud.hacktricks.xyz/pentesting-cloud/aws-security/aws-privilege-escalation/aws-lambda-privesc/ generate a path with the sequence of attacks to reach the attack goals.
	You can continue with the branch associated to a Lambda Function: https://cloud.hacktricks.xyz/pentesting-cloud/aws-security/aws-privilege-escalation/aws-lambda-privesc generate a path to reach the attack goals and concatenate the branch to the Attack Tree
	Nice, now generate the AWS CodeBuild branch of the tree visit the website: https://cloud.hacktricks.xyz/pentesting-cloud/aws-security/aws-privilege-escalation/aws-codebuild-privesc/ generate a path with the sequence of attacks to reach the attack goals, its not necessary to regenerate the entire Attack Tree, generate just the DOT code for AWS CodeBuild I will concat them manually
	Nice, now generate the AWS CodePipeline branch of the tree visit the website: https://cloud.hacktricks.xyz/pentesting-cloud/aws-security/aws-privilege-escalation/aws-codepipeline-privesc/ generate a path with the sequence of attacks to reach the attack goals, its not necessary to regenerate the entire Attack Tree, generate just the DOT code for CodePipeline I will concat them manually
Security Controls Context	I will share you the entire Attack Tree in DOT format please infer security controls for each branch. Remember that exist GuardDuty that is a native security tool in AWS
Risk context	Nice, Categorize the risk of each branch of the Attack Tree based on your historically expertise. Add a number for each branch being 1 for the most probable, 2 for the second and so on. Its not necessary to redraw the entire Attack Tree
Cosmetic details	Nice, now you will receive an Attack Tree in dot format, apply cosmetic changes: Use Arial as font, use white letters for all nodes, use #ACCFF2 for attacks nodes, use #0683FF for security control nodes

Table 3: Followed steps to compose an Attack Tree using a LLM

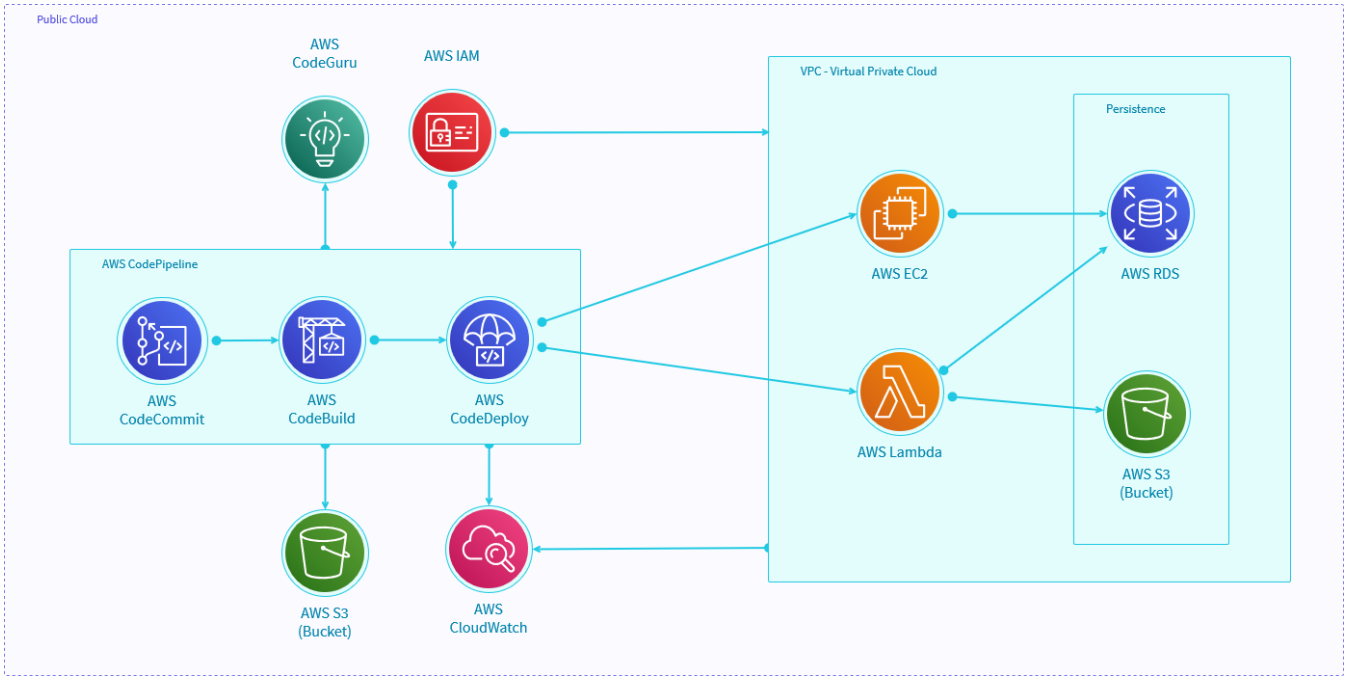


Figure 3: Topology for the proposed use case

dition will be tested with the specific SSRF attack to be run by ChaosXploit (i.e., attack to execute).

SCE experiment #2 is built from branch #1, which aims to identify whether AWS GuardDuty is able to detect privilege escalation (i.e., hypothesis) when misconfigured policies allow an attacker to use credentials stolen to gain access to another AWS services (i.e., scenario to be tested). The use of an EC2 existing profile to perform operations outside its scope could indicate an attack [47]. Once again, our assumption is that there should not be misconfigured policies, and in case there are some, GuardDuty should detect and prevent it. Eventually, that condition will be tested by launching an EC2 instance using an existing EC2 profile (i.e., attack to execute).

SCE experiment #3 is built from branch #3 and is focused on identifying whether AWS GuardDuty is able to detect lateral movement using lambda function credentials (i.e., hypothesis) when an attacker gains access to the lambda function and steals its credentials (i.e., scenario to be tested). The use of lambda credentials to perform operations outside their scope could indicate an attack [48]. Our assumption is that the lambda function policies are well configured; if not, GuarDuty should detect and prevent it. Eventually, that condition will be tested by gaining access to another AWS service using the stolen lambda credentials through an SSRF attack (i.e., attack to execute).

SCE experiment #4 is built from branch #5, which aims to determine if AWS GuardDuty is able to detect that AWS CodeBuild was used to escalate privileges (i.e., hypothesis) in situations where misconfigured IAM role assigned to CodeBuild allows an attacker to interact with another AWS services (i.e., scenario to be tested). The use of the credentials of the CodeBuild container outside its scope may indicate an attack [49]. Our assumption is that if the IAM role for CodeBuild is misconfigured, it will be detected and prevented by GuardDuty. Eventually, that condition will be tested by gaining access to the container and stealing its credentials using malicious build scripts to perform elevated IAM operations (i.e., attack to execute).

SCE #5 is built from branch #4 and is focused on identifying whether AWS CodeGuru-Security is able to detect a SQL injection in the source code (i.e., hypothesis) when an unaware developer copies generic code to perform SQL queries (i.e., scenario to be tested). A SQL injection can be used to exfiltrate a database [50]. Our assumption is that according to the documentation, AWS CodeGuru-Security has the ability to detect SQL injections. Eventually, that condition will be tested by intentionally injecting a SQL injection through ChaosXploit (i.e., attack to execute).

5.4. Running SCE Experiments #4: Validating AWS CodeBuild component

Once the security analyst has defined the SCE experiments applicable to the CSM system, he can implement them using ChaosXploit. Two of the SCE experiments described in Table 4 were implemented to demonstrate the capabilities of our proposal: SCE experiment #4 and #5.

This section describes the SCE experiment #4, designed to verify the possibility of detecting the extraction of credentials associated with an AWS CodeBuild container used in the build stage of a CI pipeline. To perform this task, it is necessary to create a specific role in IAM with the “PassRole” permission and privileges over AWS CodeBuild [49].

The execution of the attack behind SCE experiment #4 involves four key actions: i) establishing a secure tunnel to the attacker’s machine, ii) creating a repository to store the reverse

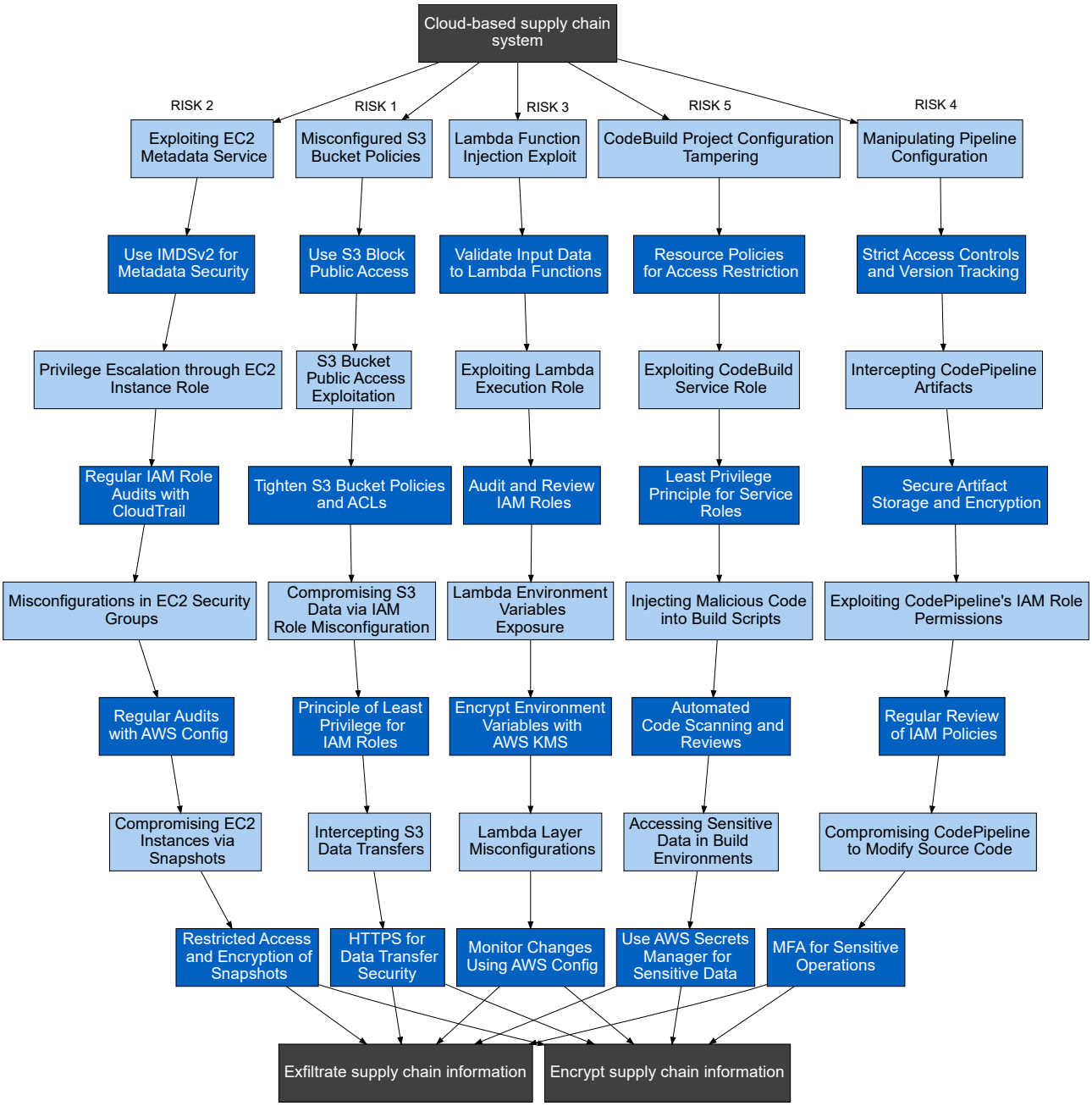


Figure 4: Specific Attack-Defense Tree generated using GPT-4. Light-blue boxes represent attack techniques. Dark-blue boxes represent security controls. Black boxes at the bottom represent attack goals.

shell command, iii) initiating a CodePipeline project, and iv) attempting to extract credentials. Each action is described in detail next.

5.4.1. *Action 1: Establishing a secure tunnel to the attacker's machine*

The first step of the SCE experiment is to establish a means of connection between the attacker's machine and the target. In this case, a secure tunnel enables communication between the attacker and the victim machine. Thus, Ngrok [51] was used to establish a TCP tunnel able to receive, through the public internet, the connection from the victim machine. Ngrok provides a temporary public URL, e.g., tcp://6.tcp.us-cal-1.ngrok.io/11576, which is utilized for the reverse shell execution (Upper terminal of Figure 5). In addition, the attacker machine must set a listening port, e.g., using Netcat, so it can receive the forwarded connection coming from Ngrok (Lower terminal of Figure 5).

SCE Experiment	Hypothesis	What to measure in the experiment	Scenario to be tested	Attack to execute
1 - S3	When AWS GuardDuty detects massive replacement operations over a S3 bucket, it preventively deny the access to the bucket.	High volume of replacement operations detected in a S3 service. Access restriction after the anomalous activity is evidenced.	A set of policies misconfigured that allows an attacker to identify S3 bucket connection and kidnap supply chain reports stored in it.	Encrypt S3 (data kidnapping) through a SSRF attack over a lambda function [46].
2 - EC2	When AWS GuardDuty detects that instance profile credentials are used to gain access to another AWS services, is possible to revoke misconfigured policies.	Events of services access using credentials from a EC2 instance. Detention of the instance after the alarm is launched.	A set of policies misconfigured that allows an attacker to launch a new EC2 instance passing an existing EC2 profile, it allows the attacker to get the privileges from existing EC2 profile.	Privilege escalation through launching an EC2 instance using an existing EC2 profile [47].
3 - Lambda	When AWS GuardDuty detects that lambda metadata credentials are used to gain access to another AWS services is possible to stop compromised service.	Events of services access using credentials from a lambda function. Detention of the lambda function after the alarm is launched.	A lambda function is implemented without sanitizing inputs it allows an attacker to get metadata credentials and gain access to another services.	Lateral movement to other services through a SSRF attack over a lambda function [48].
4 - CB	When AWS GuardDuty detects that AWS CodeBuild was used to gain elevated access by some IAM user	Events of service access using credentials from AWS CodeBuild. Detention of the container after the alarm is launched.	A misconfiguration in the IAM role of the AWS CodeBuild component that allows containers to escalate privileges across AWS environment.	Privilege escalation through stealing an AWS CodeBuild Container raw credentials [49].
5 - CP	When a developer implements a SQL injection into application's source code, AWS CodeGuru-Security identifies the issue.	Events of vulnerabilities in the CI pipeline identified by AWS CodeGuru-Security.	A developer without security awareness copying generic code to perform SQL queries on an application's database.	Database exfiltration through SQL injection attack in RDS instance [50].

Table 4: SCE experiments proposed

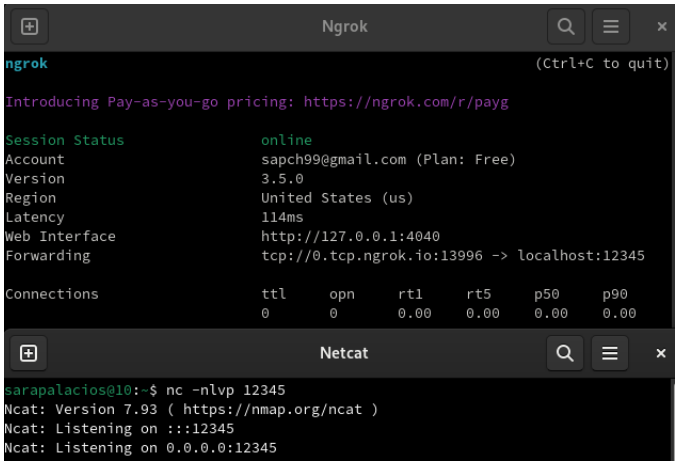


Figure 5: Creation of a tunnel with Ngrok and listening on a local port with netcat

5.4.2. *Action 2: Create a repository that stores reverse shell command*

AWS CodeBuild is a fully managed build service provided by Amazon Web Services (AWS). Such a service is designed to compile source code, run tests, and produce deployable artifacts. In this context, *buildspec.yaml* file defines the behavior of AWS CodeBuild through a set of commands and settings executed in a container, instructing how to build and package source code in a CI/CD pipeline. The *buildspec.yaml* file is stored in a repository that is associated with the CodePipeline. In this SCE experiment, the *buildspec.yaml* file is modified maliciously to inject a reverse shell at line 13, as shown in Listing 2.

```
1  phases:
2      install:
3          runtime-versions:
4              nodejs: 10
5          commands:
6              - echo "installing something"
7      pre_build:
8          commands:
9              - echo "we are in the pre build phase"
10     build:
11         commands:
12             - echo "we are in the build block"
13             - bash -c 'bash -i >& /dev/tcp/6.tcp.us-cal-1.ngrok.io/11576 0>&1' #
       REVERSE SHELL COMMAND
14             - echo "we will run some tests"
15             - grep -Fq "Cross-Site Scripting (XSS)" index.html
16     post_build:
17         commands:
18             - echo "we are in the post build phase"
19 artifacts:
20     files:
21         - '**/*'
22     name: DevOps-Code-Build
```

Listing 2: buildspec.yaml

5.4.3. *Action 3: Create a CI pipeline*

The repository containing the pivotal *buildspec.yaml* file has to be integrated into the CI/CD pipeline so the attack can be orchestrated. Once all pipeline stages are connected, the launching of the pipeline will lead to the execution of the stored reverse shell command inside the build stage. Figure 6 shows a successful reverse shell, where an attacker is connecting to an AWS CodeBuild container and listing the stored files.

5.4.4. *Action 4: Credential extraction*

The final phase involves extracting credentials from the container used by AWS CodeBuild. To extract the credentials, the `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` environment variable is utilized as it contains a URL path toward the credentials (Figure 7).

5.4.5. *Automatization of SCE experiment #4 with ChaosXploit*

Previous actions were automatized with ChaosXploit, so the hypothesis defined for this experiment can be confirmed or refuted every time the experiment is executed to validate the right

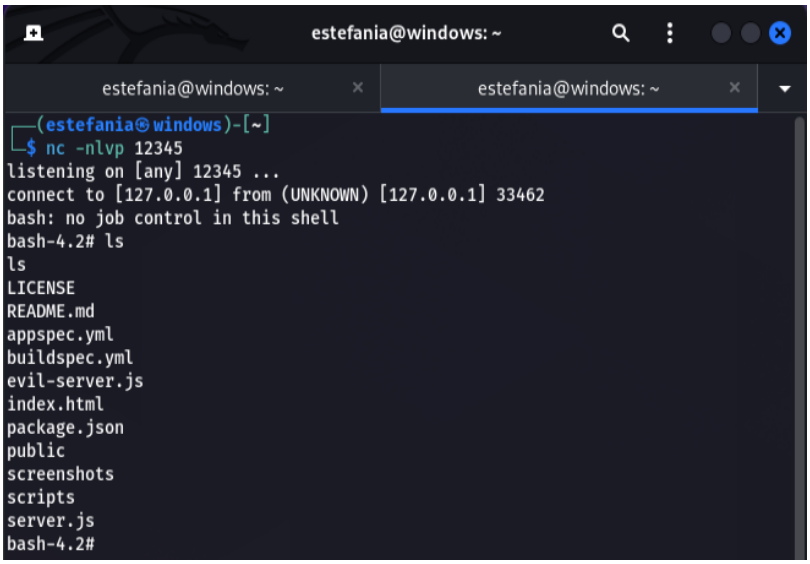


Figure 6: Reverse shell toward the AWS CodeBuild container

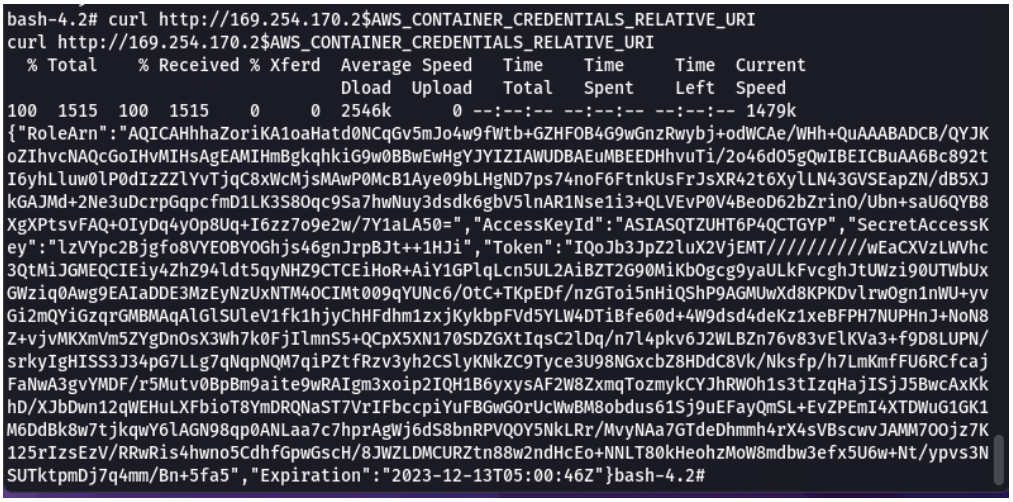


Figure 7: Extracted credentials

configuration of IAM roles and the detection capabilities of GuarDuty.

The automatization of SCE experiment #4 with ChaosXploit is done through different steps in the same order as the previous actions. Figure 8 shows the two initial steps, where the first one validates the stable state, i.e., ChaosXploit queries GuardDuty about the existence of critical or high-severity alerts. Once the stable state has been validated, the second step changes to the *buildspec.yaml* file to inject the vulnerable code. Upon the execution of these initial steps, the DevSecOps pipeline initiates.

Figure 9 shows step 3, where ChaosXploit listens on port 12345 for a connection. Once the victim is connected, ChaosXploit attempts at step 4 to extract the credentials. Finally, upon obtaining the credentials, ChaosXploit uses them in step 5 to establish a connection with the CodeBuild container and perform unauthorized administrative actions.

Thus, this SCE experiment #4 may be used by the Security Analyst to validate if GuardDuty identifies this kind of elevation privilege alert. In this case, the experiment results indicate a refutation of the hypothesis posed for this experiment in Table 4, i.e., GuardDuty would detect the use of AWS CodeBuild to gain elevated access. While the access was maintained, no GuardDuty alerts were triggered.

5.5. Running SCE Experiments #4: Validating software source code

This section describes the SCE experiment defined in row 5 of Table 4, which aims to validate whether static code analysis tools integrated into a continuous integration pipeline (CI) can identify vulnerabilities in the code implemented by developers.

The execution of this attack is composed of 4 actions: i) Creating a CI pipeline, ii) Integrating AWS CodeGuru Security, iii) Implanting a vulnerability iv) Analyzing CodeGuru Security results. The description of each action is detailed next.

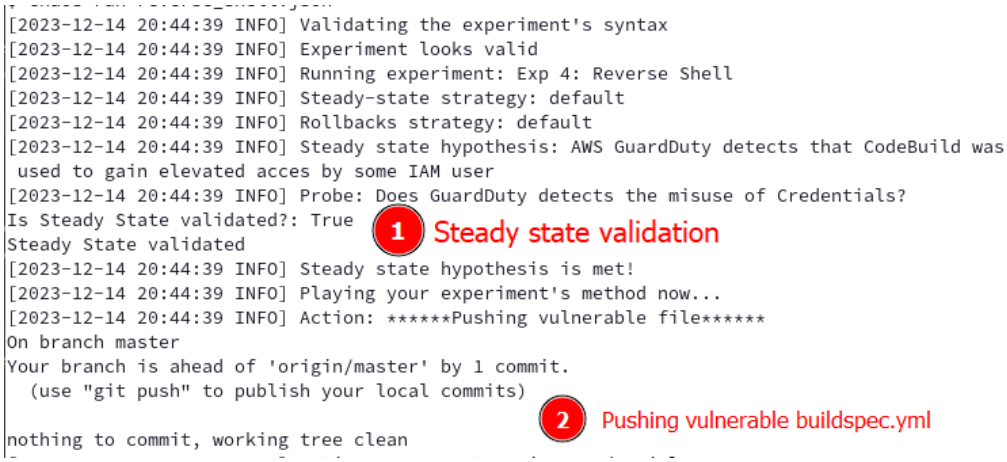


Figure 8: Automatization of SCE Experiment #4 with ChaosXploit: Steps 1 and 2

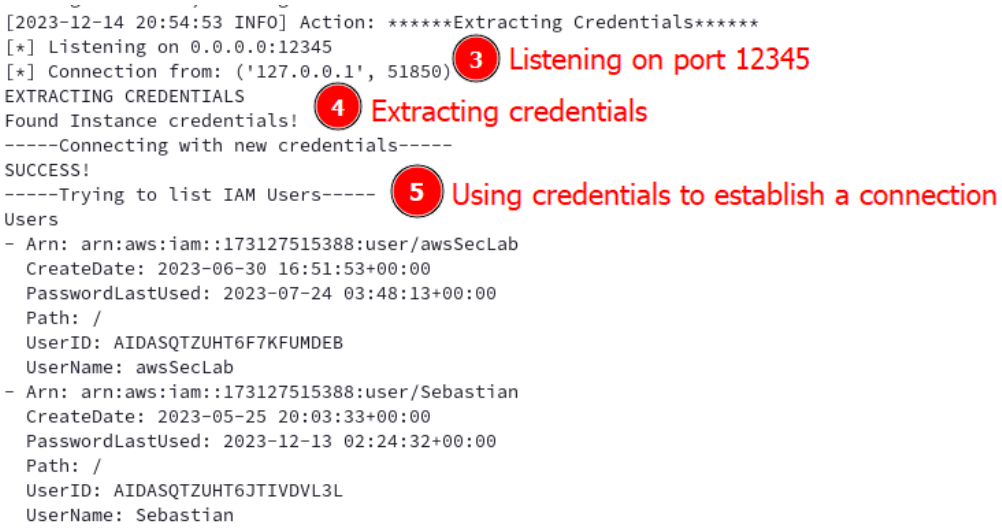


Figure 9: Automatization of SCE Experiment #4 with ChaosXploit: Steps 3, 4 and 5

5.5.1. *Action 1: Creating a CI pipeline*

To perform this task, it is necessary to create a series of roles in IAM with privileges over AWS CodeCommit, AWS CodeBuild, and AWS CodePipeline services and then assign them to the different development team members.

First, it is necessary to upload the source code to AWS CodeCommit, which is a solution that allows organizations to host secure and highly scalable private Git repositories. Subsequently, a build task must be configured using AWS CodeBuild. In this step, developers can manage the necessary aspects to create the software artifact to be deployed. Then, it will be necessary to create a pipeline using CodePipeline. This service will launch pipelines when a change in the source code is detected.

5.5.2. *Action 2: Integrating AWS CodeGuru Security*

Once the pipeline is created, it is possible to create security tasks in both CI and CD stages. In this case, we create a stage after the construction process responsible for analyzing the source code. Particularly, this approach has been chosen because auditing a compiled artifact produces fewer false positives than auditing the source code directly [52]. For this purpose, CodeGuru Security has been selected, which is an AWS service that allows to perform SAST to identify vulnerabilities in the source code (Figure 11).

5.5.3. *Action 3: Implanting the vulnerability*

In this phase, an SQL injection is intentionally added to the source code in order to identify if CodeGuru Security is able to detect it. Vulnerable source code can be found in Listing 3. This new code is committed, triggering the CI pipeline, including compilation stages and static analysis with CodeGuru.

```
def get_transaction_by_origin(id_origin):
    query = f"SELECT * FROM transactions WHERE id
     >= {id_origin};"
    conn.execute(query)
    return conn.fetchall()
```

Listing 3: Vulnerable code to SQL Injection

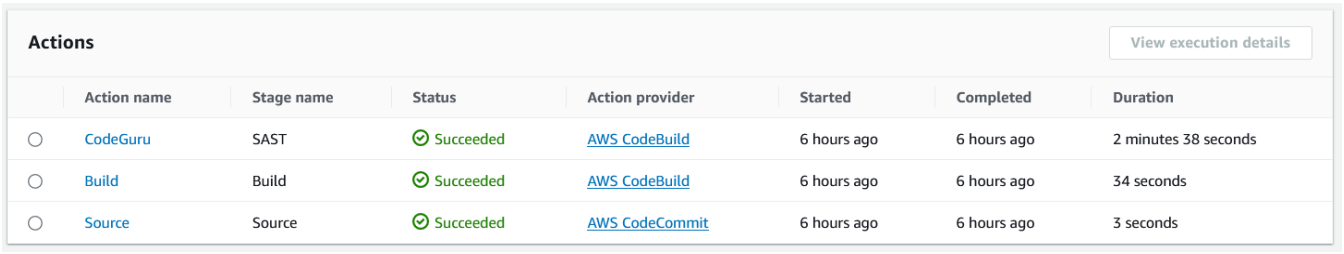


Figure 10: CI pipeline created

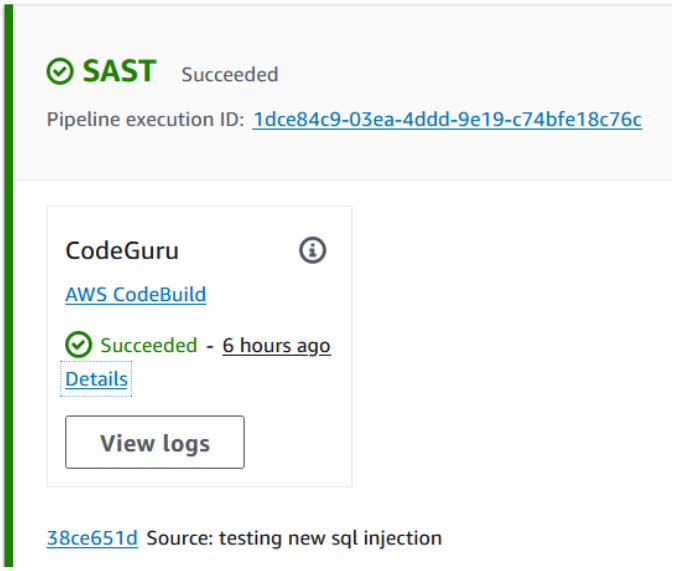


Figure 11: CodeGuru Security Integrated into CI pipeline

5.5.4. Action 4: Analyzing CodeGuru Security Results

The next phase consists of reviewing the results of CodeGuru Security through the tool's interface in order to identify if there are new vulnerabilities in the source code (Figure 12).

5.5.5. Automatization of SCE experiment #5 with ChaosXploit

Similar to experiment #4, all actions were automated using ChaosXploit in experiment #5 to verify whether the hypothesis defined for this case was correct during the experiment's execution. This is done to provide the Security Analyst with criteria to assess the effectiveness of static code analysis tools such as AWS CodeGuru.

The execution of this experiment with ChaosXploit follows the same order as shown before. Figure 13 shows the two initial steps of the experiment. Starting with the validation of the initial state, which in this case is defined by the number of high or critical vulnerabilities detected by CodeGuru. We initiate the experiment with 2 high vulnerabilities. As a second step, we address these vulnerabilities but introduce an additional one (shown in Listing 3) at the code level.

Figure 14 shows step 3, which pushes all the changes, and step 4, where ChaosXploit waits for the CI pipeline to complete the build and static analysis. Finally, a new validation of the steady state is performed, checking if AWS GuardDuty detects the new vulnerability.

In this case, this SCE experiment #5 allows the Security Analyst to demonstrate that, despite addressing the initial vulnerabilities, the new one goes undetected, leading to the experiment's failure. The hypothesis defined in Table 4 is not fulfilled as GuardDuty fails to detect the SQL injection.

6. Conclusions and future work

It is a reality that implementing security during the software life cycle is better than simply running ethical hacking tests. However, a secure development model is expensive and requires governance, technology, and people. In this scenario, the DevSecOps paradigm permits to build and provide secure software quickly and agilely, allowing companies to generate value and keep their systems secure at a lower cost, increasing their ROI.

Moreover, the incorporation of artificial intelligence holds the potential to seamlessly integrate and automate various security processes within a DevOps framework, thus facilitating its progressive advancement. Examples of this promising integration between AI and DevOps encompass boosting the creation of threat models enhancing the precision in pinpointing vulnerabilities within the source code, the operational application, and its constituent elements. Furthermore, artificial intelligence contributes to proposing robust control measures to

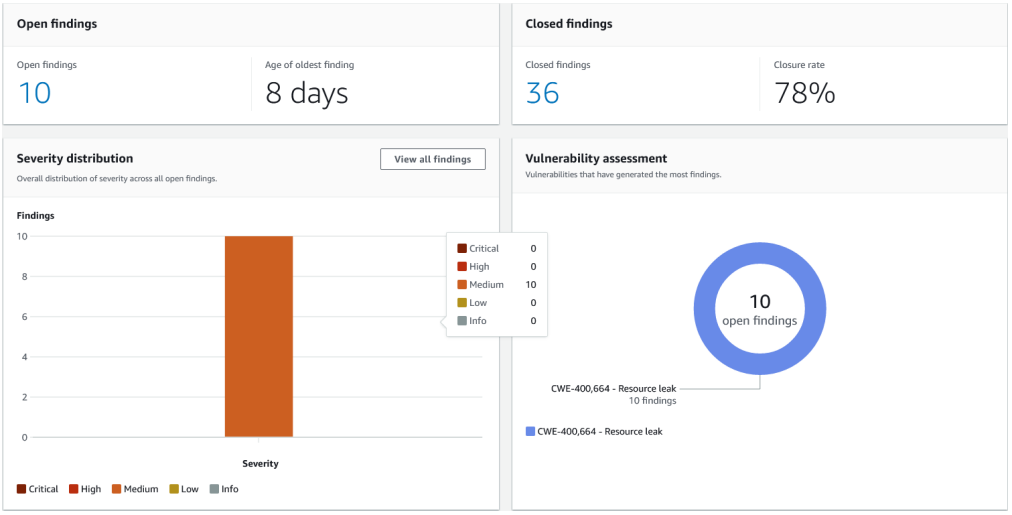


Figure 12: Results of CodeGuru Security

```
[2023-12-14 16:41:21 INFO] Validating the experiment's syntax
[2023-12-14 16:41:21 INFO] Experiment looks valid
[2023-12-14 16:41:21 INFO] Running experiment: Exp 5: CodeGuru
[2023-12-14 16:41:21 INFO] Steady-state strategy: default
[2023-12-14 16:41:21 INFO] Rollbacks strategy: default
[2023-12-14 16:41:21 INFO] Steady state hypothesis: When a developer implements a SQL in
jection into application's source code AWS GuardDuty- Security identifies the issue.
[2023-12-14 16:41:21 INFO] Probe: Do I have Critical or high vulnerabilities?
Starting code guru session
Creating Session
Last scan created at: 2023-12-14 16:38:59.254000-05:00
{'critical': 0.0, 'high': 2.0, 'info': 0.0, 'low': 0.0, 'medium': 10.0}
There are critical or high issues
[2023-12-14 16:41:29 INFO] Steady state hypothesis is met!
[2023-12-14 16:41:29 INFO] Playing your experiment's method now...
[2023-12-14 16:41:29 INFO] Action: ******Creating vulnerable file******
```

1 Steady State Validation

2 Fixing issues and adding SQL Injection

Figure 13: Automatization of SCE Experiment #5 with ChaosXploit steps 1 and 2

```
[2023-12-14 16:41:29 INFO] Action: ******Pushing vulnerable file******
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
Waiting for pipeline to end
[2023-12-14 16:46:33 INFO] Steady state hypothesis: When a developer implements a SQL in
jection into application's source code AWS GuardDuty- Security identifies the issue.
[2023-12-14 16:46:33 INFO] Probe: Do I have Critical or high vulnerabilities?
Starting code guru session
Creating Session
Last scan created at: 2023-12-14 16:43:14.324000-05:00
{'critical': 0.0, 'high': 0.0, 'info': 0.0, 'low': 0.0, 'medium': 10.0}
Didn't find any critical or high issues
[2023-12-14 16:46:42 CRITICAL] Steady state probe 'Do I have Critical or high vulnerabil
ities?' is not in the given tolerance so failing this experiment
[2023-12-14 16:46:42 INFO] Experiment ended with status: deviated
[2023-12-14 16:46:42 INFO] The steady-state has deviated, a weakness may have been disco
vered
```

3 Pushing changes

4 Validating if codeGuru detects the new vulnerability

Figure 14: Automatization of SCE Experiment #5 with ChaosXploit steps 3 and 4

fortify both the application and its underlying infrastructure.

This paper presented an innovative way to advance a DevSecOps practice by using LLMs to automate the creation of Attack-Defense Trees, which is, at the same time, the input to define SCE experiments applicable to both the system and the CI/CD pipelines. This proposal constitutes an additional layer of security to SAST, SCA, and DAST tools, which may be integrated into a CI/CD pipeline. As far as we know, there is no similar proposal that integrates LLMs and SCE to enhance cybersecurity in a software supply chain.

The integration of LLMs and SCE within a software supply chain offers effective advantages. Firstly, automating the construction of an attack model significantly reduces the time burden on Security Analysts within agile development teams. Secondly, SCE introduces a crucial capability to uncover vulnerabilities that might elude detection by conventional security tools integrated into a DevSecOps practice, as demonstrated in Section 5. This becomes particularly valuable when SCE experiments are seamlessly integrated into CI/CD pipelines. This dual benefit aligns with the imperative of swiftly and securely building systems within corporate environments.

As future work, we plan to compare the output of other LLMs to generate Attack-Defense Trees. Also, we intend to propose new SCE experiments applicable to DevSecOps practices and to propose new ways to automate the discovery of vulnerabilities in the business logic of systems.

Acknowledgment

This work has been partially supported by the Spanish Ministry of Universities linked to the European Union through the NextGenerationEU program under Margarita Salas postdoctoral fellowship (172/MSJD/22) and from the strategic project CDL-TALENTUM from the Spanish National Institute of Cybersecurity (INCIBE) and by the Recovery, Transformation and Resilience Plan.

References

[1] M. Paquet-Clouston, S. García, On the motivations and challenges of affiliates involved in cybercrime, Trends in Organized Crime (Dec 2022). doi:10.1007/s12117-022-09474-x.
URL https://doi.org/10.1007/s12117-022-09474-x

[2] J. Simon, A. Omar, Cybersecurity investments in the supply chain: Coordination and a strategic attacker, European Journal of Operational Research 282 (1) (2020) 161–171. doi:https://doi.org/10.1016/j.ejor.2019.09.017.

[3] R. Alkhadra, J. Abuzaid, M. AlShammari, N. Mohammad, Solar winds hack: In-depth analysis and countermeasures, in: 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT), 2021, pp. 1–7. doi:10.1109/ICCCNT51525.2021.9579611.

[4] A. Almogahed, M. Omar, N. H. Zakaria, A. Alawadhi, Software security measurements: A survey, in: 2022 International Conference on Intelligent Technology, System and Service for Internet of Everything (ITSS-IoE), 2022, pp. 1–6. doi:10.1109/ITSS-IoE56359.2022.9990968.

[5] D. Geer, E. Jardine, E. Leverett, On market concentration and cybersecurity risk, Journal of Cyber Policy 5 (1) (2020) 9–29. doi:10.1080/23738871.2020.1728355.

[6] Z. Shen, S. Chen, A survey of automatic software vulnerability detection, program repair, and defect prediction techniques, Security and Communication Networks 2020 (2020) 8858010. doi:10.1155/2020/8858010.

[7] A. Basiri, N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds, C. Rosenthal, Chaos engineering, IEEE Software 33 (3) (2016) 35–41. doi:10.1109/MS.2016.60.

[8] X. Ramaj, R. Colomo-Palacios, M. Sánchez-Gordón, V. Gkioulos, Towards a DevSecOps-Enabled Framework for Risk Management of Critical Infrastructures, in: M. Yilmaz, P. Clarke, A. Riel, R. Messnarz (Eds.), Systems, Software and Services Process Improvement, Springer Nature Switzerland, Cham, 2023, pp. 47–58.

[9] K. Shortridge, A. Rinehart, Security Chaos Engineering: Sustaining Resilience in Software and Systems, O'Reilly Media, 2023.

[10] M. Bedoya, S. Palacios, D. Diaz-López, P. Nespoli, E. Laverde, S. 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, Association for Computing Machinery, New York, NY, USA, 2023. doi:10.1145/3600160.3605076.

[11] A. Rinehart, K. Shortridge, Security Chaos Engineering Gaining Confidence in Resilience and Safety at Speed and Scale, Tech. rep. (2021).

[12] U. Koc, P. Saadatpanah, J. S. Foster, A. 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, Association for Computing Machinery, New York, NY, USA, 2017, p. 35–42. doi:10.1145/3088525.3088675.

[13] M. Cankar, N. Petrovic, J. Pita Costa, A. Cernivec, J. Antic, T. Martincic, D. 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, Association for Computing Machinery, New York, NY, USA, 2023, p. 201–205. doi:10.1145/3578245.3584943.
URL https://doi.org/10.1145/3578245.3584943

[14] M. Al-Hawawreh, A. Aljuhani, Y. Jararweh, ChatGPT for cybersecurity: practical applications, challenges, and future directions, Cluster Computing 26 (6) (2023) 3421–3436. doi:10.1007/s10586-023-04124-5.

[15] A. Nguyen-Duc, B. Cabrero-Daniel, A. Przybylek, C. Arora, D. Khanna, T. Herda, U. Rafiq, J. Melegati, E. Guerra, K.-K. Kemell, M. Saari, Z. Zhang, H. Le, T. Quan, P. Abrahamsson, Generative artificial intelligence for software engineering – a research agenda (2023). arXiv:2310.18648.

[16] M. Gupta, C. Akiri, K. Aryal, E. Parker, L. Praharaj, From ChatGPT to ThreatGPT: Impact of Generative AI in Cybersecurity and Privacy, IEEE Access 11 (2023) 80218–80245. doi:10.1109/ACCESS.2023.3300381.

[17] Z. Szabó, V. Bilicki, A New Approach to Web Application Security: Utilizing GPT Language Models for Source Code Inspection, Future Internet 15 (10) (2023). doi:10.3390/fi15100326.

[18] M. Nair, R. Sadhukhan, D. Mukhopadhyay, How Hardened is Your Hardware? Guiding ChatGPT to Generate Secure Hardware Resistant to CWEs, in: S. Dolev, E. Gudes, P. Paillier (Eds.), Cyber Security, Cryptology, and Machine Learning, Springer Nature Switzerland, Cham, 2023, pp. 320–336.

[19] T. McIntosh, T. Liu, T. Susnjak, H. Alavizadeh, A. Ng, R. Nowrozy, P. Watters, Harnessing GPT-4 for generation of cybersecurity GRC policies: A focus on ransomware attack mitigation, Computers & Security 134 (2023) 103424. doi:https://doi.org/10.1016/j.cose.2023.103424.

[20] O. Gadyatskaya, D. Papuc, ChatGPT Knows Your Attacks: Synthesizing Attack Trees Using LLMs, in: C. Anutariya, M. M. Bonsangue (Eds.), Data Science and Artificial Intelligence, Springer Nature Singapore, Singapore, 2023, pp. 245–260.

[21] W. Wideł, M. Audinot, B. Fila, S. Pinchinat, Beyond 2014: Formal Methods for Attack Tree–Based Security Modeling, ACM Comput. Surv. 52 (4) (aug 2019). doi:10.1145/3331524.

[22] Optum, Chaoslingr: Introducing security into chaos testing, https://github.com/Optum/ChaoSlingr, last time accessed: 2022-11-9 (April 2019).

[23] C. Konstantinou, G. Stergiopoulos, M. Parvania, P. Esteves-Verissimo, Chaos Engineering for Enhanced Resilience of Cyber-Physical Systems, in: 2021 Resilience Week (RWS), 2021, pp. 1–10. doi:10.1109/RWS52686.2021.9611797.

[24] J. Downs, E. Vogel, A plant-wide industrial process control problem, Computers & Chemical Engineering 17 (3) (1993) 245–255, Industrial challenge problems in process control. doi:https://doi.org/10.1016/0098-1354(93)80018-I.

[25] K. A. Torkura, M. I. Sukmana, F. Cheng, C. Meinel, CloudStrike: Chaos Engineering for Security and Resiliency in Cloud Infrastructure, IEEE Access 8 (2020) 123044–123060. doi:10.1109/ACCESS.2020.3007338.

[26] K. A. Torkura, M. Sukmana, F. Cheng, C. Meinel, Continuous auditing and threat detection in multi-cloud infrastructure, Computers and Security 102 (2021) 102124. doi:10.1016/j.cose.2020.102124.

[27] S. Sharieh, A. Ferworn, Securing APIs and Chaos Engineering, in: 2021 IEEE Conference on Communications and Network Security (CNS), 2021, pp. 290–294. doi:10.1109/CNS53000.2021.9705049.

[28] T. Bailey, P. Marchione, P. Swartz, R. Salih, M. Clark, R. Denz, Measuring resiliency of system of systems using chaos engineering experiments, in: 2022 SPIE 12117, Disruptive Technologies in Information Sciences VI, Vol. 1211704, 2022, p. 26. doi:10.1117/12.2632779.

[29] K. Shortridge, From Lemons to Peaches: Improving Security ROI through Security Chaos Engineering, in: 2022 IEEE Secure Development Conference (SecDev), 2022, pp. 59–60. doi:10.1109/SecDev53368.2022.00021.

[30] S. Palacios Chavarro, P. Nespoli, D. Díaz-López, Y. 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). doi:10.3390/bdcc7010001.

[31] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: Neural Information Processing Systems, 2017.
URL https://api.semanticscholar.org/CorpusID:13756489

[32] M. Alawida, S. Mejri, A. Mehmood, B. Chikhaoui, O. Isaac Abiodun, A comprehensive study of chatgpt: Advancements, limitations, and ethical considerations in natural language processing and cybersecurity, Information 14 (8) (2023). doi:10.3390/info14080462.

[33] P. Nespoli, D. Papamartzivanos, F. G. Marmol, G. Kambourakis, Optimal countermeasures selection against cyber attacks: A comprehensive survey on reaction frameworks, IEEE Communications Surveys Tutorials 20 (2) (2018) 1361–1396. doi:https://doi.org/10.1109/COMST.2017.2781126.

[34] S. Cheung, U. Lindqvist, M. W. Fong, Modeling multistep cyber attacks for scenario recognition, Proceedings DARPA Information Survivability Conference and Exposition 1 (2003) 284–292 vol.1.
URL https://api.semanticscholar.org/CorpusID:18650610

[35] M. S. Haque, An evolutionary approach of attack graphs and attack trees: A survey of attack modeling, 2017.

[36] B. Kordy, L. Piètre-Cambacédès, P. Schweitzer, Dag-based attack and defense modeling: Don't miss the forest for the attack trees, Computer Science Review 13-14 (2014) 1–38. doi:https://doi.org/10.1016/j.cosrev.2014.07.001.

[37] V. Saini, Q. Duan, V. Paruchuri, Threat modeling using attack trees, Journal of Computing Sciences in Colleges 23 (04 2008).

[38] K. Edge, G. Dalton, R. Raines, R. Mills, Using attack and protection trees to analyze threats and defenses to homeland security, 2006, pp. 1 – 7. doi:10.1109/MILCOM.2006.302512.

[39] S. Bistarelli, F. Fioravanti, P. Peretti, Defense trees for economic evaluation of security investments, Vol. 2006, 2006, p. 8 pp. doi:10.1109/ARES.2006.46.

[40] R. Kumar, R. Goyal, On cloud security requirements, threats, vulnerabilities and countermeasures: A survey, Computer Science Review 33 (2019) 1–48. doi:https://doi.org/10.1016/j.cosrev.2019.05.002.

[41] OWASP Application Security Verfication Standard (2023).
URL https://owasp.org/www-project-application-security-\
\verification-standard/

[42] A. Jøsang, M. Ødegaard, E. Oftedal, Cybersecurity through secure software development, in: M. Bishop, N. Miloslavskaya, M. Theocharidou (Eds.), Information Security Education Across the Curriculum, Springer International Publishing, Cham, 2015, pp. 53–63.

[43] S. Al-Saqqa, S. Sawalha, H. Abdelnabi, Agile software development: Methodologies and trends, Int. J. Interact. Mob. Technol. 14 (2020) 246–270.
URL https://api.semanticscholar.org/CorpusID:225548331

[44] H. S. Lallie, K. Debattista, J. Bal, A review of attack graph and attack tree visual syntax in cyber security, Computer Science Review 35 (2020) 100219. doi:https://doi.org/10.1016/j.cosrev.2019.100219.

[45] Incorporating business logic to get the best out of DAST (2022).
URL https://www.invicti.com/blog/docs-and-faqs/incorporate-business-logic-get-the-best-out-of-dast/

[46] AWS, The anatomy of ransomware event targeting data residing in Amazon S3, https://aws.amazon.com/es/blogs/security/anatomy-of-a-ransomware-event-targeting-data-in-amazon-s3/, last time accessed: 2023-06-31 (February 2023).

[47] N. Dahan, Auditing IAM PassRole: A Problematic Privilege Escalation Permission, https://ermetic.com/blog/aws/auditing-passrole-a-problematic-privilege-escalation-permission/, last time accessed: 2023-06-31 (January 2021).

[48] H. T. Cloud, Steal IAM Credentials and Event Data from Lambda, https://hackingthe.cloud/aws/exploitation/lambda-steal-iam-credentials/, last time accessed: 2023-06-31 (February 2023).

[49] C. Polop, Aws - codebuild privesc - hacktricks cloud.
URL https://cloud.hacktricks.xyz/pentesting-cloud/aws-security/aws-privilege-escalation/aws-codebuild-privesc

[50] C. Polop, SQL injection.
URL https://book.hacktricks.xyz/pentesting-web/sql-injection

[51] Ngrok - Secure introspectable tunnels to localhost, https://ngrok.com/, accessed: December 2023.

[52] Source Code Analyzer (2023).
URL https://www.veracode.com/security/source-code-security-analyzer

0.4.4 Paper presented at IEEE Computing magazine

IEEE Internet Computing is a magazine in computer science and the Internet that seeks to disseminate efforts and experiences in key trends in Internet technologies and applications. The magazine has a multidisciplinary emphasis on different technology branches, including security, privacy, web applications, and other topics. More information about IEEE Internet Computing Magazine may be found at `https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=4236`

This article summarizes part of the research of this master thesis where SCE experiments applicable to a DevSecOps environment are demonstrated easily and understandably for all the actors of a software process.

Applying Chaos to reach Security: On The Application of Security Chaos Engineering to the DevSecOps practice

Martin Bedoya, *School of Engineering, Science and Technology, Universidad del Rosario, Bogotá, 111711, Colombia*

Sara Palacios, *School of Engineering, Science and Technology, Universidad del Rosario, Bogotá, 111711, Colombia*

Daniel Díaz-López, *School of Engineering, Science and Technology, Universidad del Rosario, Bogotá, 111711, Colombia. Tandon School of Engineering, New York University, Brooklyn, 11201, US*

Pantaleone Nespoli*, *Dept. of Information and Communications Engineering, University of Murcia, 30100, Spain*
Corresponding author: Pantaleone Nespoli (pantaleone.nespoli@um.es)

Abstract—System security is a big challenge for many organizations, especially when software complexity increases. It is important to integrate new methodologies that support the finding of system errors, orientate system designs, and contribute to building holistic security. This research explores the integration of Security Chaos Engineering (SCE) within the DevSecOps lifecycle, aiming to enhance the security posture of software development. By leveraging this integration, it is possible to conduct proactive SCE experiments at the early stages of software development. Automation plays a pivotal role, with a predefined set of actions facilitating seamless integration into Continuous Integration/Continuous Deployment (CI/CD) pipelines. To demonstrate the feasibility of such integration, a Cloud-based use case is proposed, in which SCE is able to proactively improve the infrastructure's security posture. The study emphasizes the complementarity of SCE with traditional security testing approaches, highlighting its ability to address specific risk scenarios.

Why traditional cybersecurity strategies fall?

Undoubtedly, cybersecurity has become an increasingly pressing issue for organizations across all sectors in recent years. This issue has been evidenced through different cybersecurity incidents from high-profile data breaches and ransomware attacks to insider threats and supply chain attacks. Thus, the risks to which the digital ecosystems are exposed have never been more prominent, and despite the struggles of academia and industry to develop novel defensive strategies against such threats, cybercriminals continue finding innovative ways to bypass them.

Traditionally, cybersecurity's duty has been to build robust defenses to protect organizations against known threats and vulnerabilities in a proactive fashion. The existence of those defenses certainly brings a sense of security, and many cybersecurity strategies may even assume mistakenly that such security measures are flawless. It is the case of Equifax, a company that experienced a massive cyberattack that resulted in the unauthorized access and theft of sensitive personal information of approximately 147 million individuals [1]. Investigations revealed several vulnerabilities and shortcomings in their security practices, as the breach was primarily attributed to a failure in implementing security patches for a known vulnerability in web application framework.

In recent years, a novel methodology applicable to secure computer systems has emerged, which is known as Security Chaos Engineering (SCE). SCE is based on experiments designed to identify failures in security controls that might go unnoticed using traditional security testing methodologies. The outcomes produced by SCE experiments trigger a feedback loop that informs enhancements to both system design and operation [2].

This article argues that SCE can be significantly

XXXX-XXX © 2024 IEEE
Digital Object Identifier 10.1109/XXX.0000.0000000

applied to DevSecOps methodology, providing outstanding improvements in terms of overall security. To this extent, DevSecOps methodology [3] can be described as a series of practices that aim to speed up the software development life cycle and ensure proper security compliance. Nowadays, it is more evident that a DevSecOps practice needs to integrate security components in the Continuous Delivery (CI) and Continuous Deployment (CD) pipelines to achieve a security audit at early stages. Additionally, an early integration of security allows to detect and remediate vulnerabilities in systems before being deployed in production. In such a traditional DevSecOps scenario, ethical hacking tests are still needed for two fundamental reasons: i) application flows can be too complex for automatic Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) tools, and ii) robots can not test business logic.

Even if SAST and DAST security tests are still fundamental to validate the security posture of a system, SCE introduces a proactive dimension to the security testing paradigm. SCE deliberately injects controlled chaos into the system, simulating real-world attack scenarios. This approach provides a unique advantage by stress-testing security controls and response mechanisms in a live environment. By leveraging SCE-DevSecOps integration, organizations can achieve a more comprehensive and realistic assessment of their security posture, enhancing their ability to detect, mitigate, and adapt to evolving threats throughout the software development lifecycle.

Where chaos came from?

CE methodology first emerged in 2011 at Netflix to strengthen the system's resiliency and improve performance. In the same year, Netflix released a tool named ChaosMonkey, created to test the platform's stability by injecting random faults on the internal instances [4]. A year later, Netflix added new fault modes that were reported on a new tool named SimianArmy.

In its origins, CE mainly focused on validating the system's resilience. Later, Rinehart and Shortridge introduced the SCE methodology [5], [6] realizing that traditional security testing methods, such as penetration testing and vulnerability scanning, did not necessarily reflect the complexity of modern systems and the unpredictability of real-world attacks. To address this gap, the authors proposed a fresh approach that applied the principles of CE to digital security, which involved injecting intentionally security vulnerabilities into systems to see how security teams would detect and handle them. At the moment, industry and academia agree that SCE can be defined as "the identification of security control failures through proactive experimentation to build confidence in the capability of a system to defend against malicious conditions in production" [5].

Together with the popularization of the SCE methodology, some tools and frameworks emerged to support this approach, including ChaosSlingr [7], CloudStrike [8] and Prowler Pro [9].

In 2022, Palacios *et al.* presented ChaosXploit [10], an SCE-powered framework that employs a knowledge database composed of attack trees to expose vulnerabilities that exist in a software solution that has been previously defined as a target. Within the authors' proposal, different experiments were described and executed to validate the feasibility of such a tool in terms of auditing the security of cloud-managed services.

Thus, ChaosXploit allowed comprehensive and semi-automated tests over information systems. One important component that leads the SCE experiments in ChaosXploit is an attack tree, which represents the possible branches, i.e., paths, that an attacker could follow to achieve a specific attack goal.

As previously mentioned, SCE helps to identify security gaps, which are often overlooked by traditional assessment methods. Some of these security gaps may be related to software vulnerabilities, misconfigurations, and logical flaws, among others. In addition, SCE enhances the system audit process by providing a set of observed variables that help explain the behavior of a system.

At their core, CE and SCE share a common methodology while differentiating in the final objective. That is, SCE follows the scientific method as its underlying framework, drawing on the principles of chaos engineering. While these principles outline the methodology followed by CE, its approach changes when extended to security. The steps followed in the SCE methodology are detailed next:

1) Define the system behavior and establish evaluation metrics, which implies improving the observability of the system, gaining insights, detecting hidden patterns, and identifying security flaws.
2) Identify the system steady-state, which is critical to understand the system equilibrium and ensure explainability over the experiment.
3) Set a hypothesis to be validated, which led to focused experimentation.
4) Run the experiment in a secure environment, incorporating real-life events, i.e., controlled chaos, which enables a comprehensive security assessment.

On the other hand, it is known DevSecOps prac-

tices aim to improve the security of the software lifecycle. For this purpose, automatic build and deployment are essential as they contribute to reduce delivery times and ensures that all stakeholders are focused on their roles. By integrating cybersecurity objectives into Continuous Integration (CI) and Continuous Deployment (CD) pipelines, DevSecOps practitioners aim to guarantee that software is secure and business data are protected. Different strategies may be used in the application of DevSecOps practices, however, such strategies are far from detecting all the software vulnerabilities, having as a consequence an enormous amount of published vulnerable software.

So, how does SCE contribute to DevSecOps? SCE offers a clear contribution since it enable a proactive threat modeling that allows spotting hard-to-detect vulnerabilities in the IT operation and development domains. Flaws that may be detected by SCE in the IT operation domain include component misconfigurations, capacity shortages, and uncontrolled assets, among others. On the other hand, flaws that may be detected by SCE in the development domain include flaws in the business logic, exploitable code, lack of security stories, exposed containers, and insecure API access control, among others.

Securing DevOps Through Controlled Chaos

To illustrate the potential enhancement of DevOps through the SCE paradigm, let us contemplate a practical scenario. Imagine an e-commerce company that needs updates in its platforms to address technological obsolescence. To avoid service disruptions during this update, the company opts to refactor its e-commerce modules and transition them to a cloud-based environment. Following the successful migration, the company envisions the development of additional functionalities to enrich the customer experience, requiring a CD approach. In this strategic evolution, security is a requirement, thus Amazon Web Services (AWS) is chosen as cloud provider due to its robust DevSecOps infrastructure, aligning seamlessly with the business objectives.

In this scenario, it is worth remarking that a DevSecOps infrastructure requires different components, including: i) source code repositories, e.g., AWS CodeCommit, ii) modules to build/compile code, e.g., AWS CodeBuild, iii) modules to deploy a compiled code in a production environment, e.g., AWS CodeDeploy, iv) tools to scan source code, e.g., AWS CodeGuru, v) modules that automate the entire process, e.g., AWS CodePipeline.

The above DevSecOps components must be integrated with the components of the solution architecture, which have been defined by the company, to coexist in the cloud. In this use case, we can define a simple topology composed of: i) an AWS ECS container to serve the e-commerce front-end, ii) an AWS EC2 instance to handle web requests, iii) an RDS database to store transactions, and iv) an AWS Lambda function to run background tasks.

From these components, an Attack-Defense Tree must be created by analyzing the tactics, techniques, and procedures employed by attackers to affect any cloud components employed in the use case. For our use case, a tree was created manually (Figure 1), which contains offenses against AWS CodeBuild and CodePipeline, based on known attack procedures contained in HackTricks [11].

The Attack-Defense Tree shown in Figure 1 consists of three types of nodes, specifically: i) light blue nodes representing actions taken by an attacker to exploit a vulnerability, ii) dark blue nodes representing security controls that can be implemented to mitigate the attack, and lastly iii) gray nodes representing attack goals pursued by a potential malicious actor.

As one may note, the Attack-Defense Tree offers a great opportunity to evaluate the defensive capabilities of a system, especially hypothesizing a scenario where all the offensive actions (light blue boxes) are possible and successful, i.e., the security controls fail in every step of the attack. One could argue that this assumption might be considered exaggerated and improbable in typical scenarios. Nevertheless, in certain organizational contexts, a persistent and motivated threat actor could surpass conventional probabilities.

Thus, for our use case the security test will be focused on validating the countermeasures (dark blue boxes) that exist in the Attack-Defense Tree. The countermeasures are essentially security settings in already operational cloud services, like AWS EC2, or added cybersecurity services, like CodeGuru.

Furthermore, the constructed Attack-Defense Tree depicts two branches that an insider attacker, who is part of the company and has certain privileges over the development environment, could follow to achieve his malicious goals. The first main branch (left-hand) represents a scenario where we suppose an insider attacker identifies that he has permission to self-promote source code and change the AWS CodePipeline stages, so he may conduct two types of attacks (sub-branches):

- Manipulate the application's source code before it is deployed to introduce a SQL injection vulner-

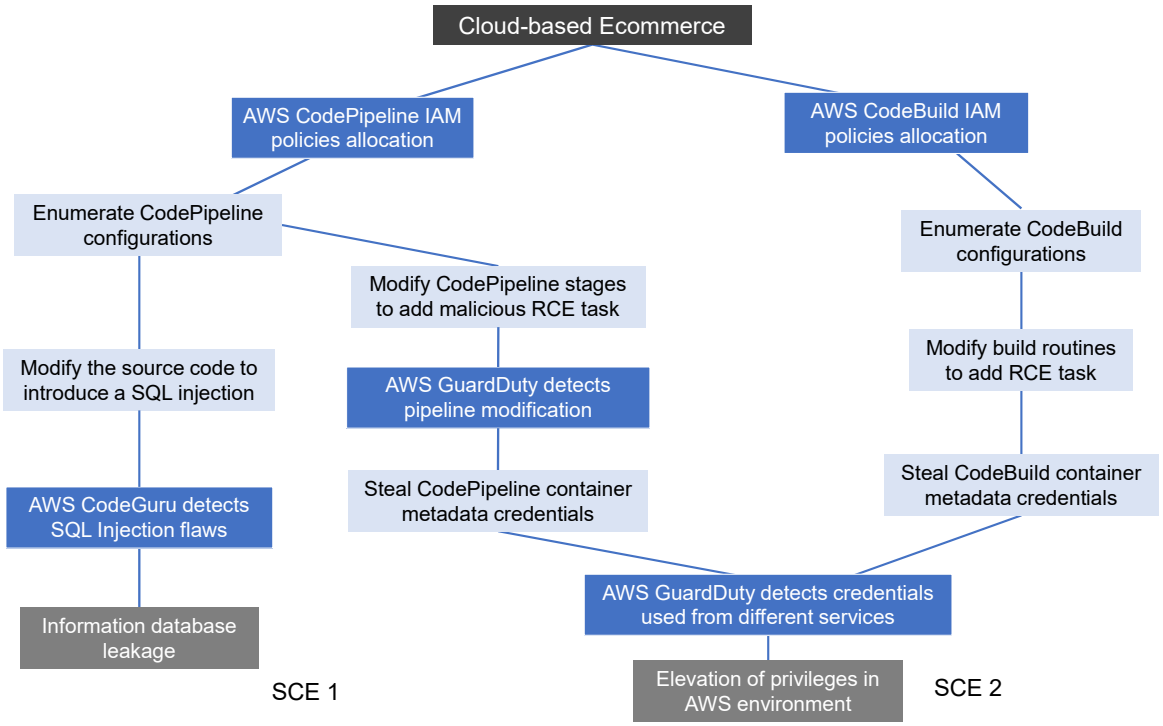


FIGURE 1: Attack-Defense Tree designed for the proposed Cloud-based use case

ability that will allow him to exfiltrate information in the future.

- Add an additional stage in the pipeline that will allow him to gain access to the CodePipeline container and then extract credentials and escalate privileges.

The second main branch (right-hand) is related to the AWS CodeBuild component, which allows building software artifacts. In this branch we suppose the attacker identifies a misconfigured permission for this component (IAM:PassRole), manipulate the compilation routine (defined in buildspec.yml) to include malicious code that will allow him access to the AWS CodeBuild container to extract its metadata credentials, and escalate privileges in the entire AWS environment.

From both branches, it is possible to identify security controls that could help mitigate or disrupt the attack (dark blue boxes). In the first branch, a preventive security control that would have avoided the attack would be a set of properly configured policies that do not allow a developer to self-promote code without a peer review. If the previous control fails for the first branch, AWS CodeGuru could be employed as detective control, as it may analyze the source code of an application to detect code vulnerabilities and has support for languages such as Java, .NET, Python, and JavaScript, among others. CodeGuru is integrated natively with AWS CodePipeline, being able to execute SAST on the code that will be deployed.

Regarding the second branch, the main preventive control would be to configure correctly the policies for the AWS CodeBuild component which would avoid the use of the IAM:PassRole permission. Concretely, the IAM:PassRole permission allows his holder to designate an IAM role to an AWS component, granting privileges over other components. If previous control fails, AWS GuardDuty could be employed as detective control, as it reviews the logs of all environment components in search of anomalous behavior. Particularly, GuardDuty has the ability to detect that credentials associated with a component are being used by other

SCE Experiment	Hypothesis	What to measure in the experiment	Scenario to be tested	Attack to execute
1	When a developer implements a SQL injection into application's source code, AWS CodeGuru-Security identifies the issue.	Events of vulnerabilities in the CI pipeline identified by AWS CodeGuru-Security.	A developer without security awareness copying generic code to perform SQL queries on an application's database.	Database exfiltration through SQL injection attack in RDS instance.
2	When metadata credentials are used to gain access to another service AWS GuardDuty generates a security alarm	Events of service access using credentials from AWS CodeBuild. Detention of the container after the alarm is launched.	A misconfiguration in the IAM role of the AWS CodeBuild component that allows containers to escalate privileges across AWS environment.	Privilege escalation through stealing an AWS CodeBuild Container raw credentials.

TABLE 1: SCE experiments proposed for the Cloud-based use case

AWS components.

Once the Attack-Defense Tree has been constructed, it is possible to propose SCE experiments for each branch, which follow the CE methodology described in the previous section. Table 1 presents a summary of 2 SCE experiments, detailing the hypothesis to be validated, parameters to be measured in the experiment, scenario to be tested, and attack to execute.

Specifically, experiments #1 and #2 aim to validate the detective and corrective controls that the organization will have in case an insider attacker injects vulnerable code into the application's source code, or steal credentials from the CodeBuild container, respectively.

For the mentioned scenario, SCE experiment #1 will be developed, as it is ideal for showcasing the effectiveness of SCE in a DevSecOps environment. This experiment aims to validate the effective functionality of a security tool, i.e., CodeGuru, to detect vulnerabilities in the source code introduced by an insider attacker. The metric for this experiment revolves around the number of high or critical vulnerabilities identified by CodeGuru. During the execution of this experiment, ChaosXploit was configured to perform the following actions:

1) Execute CodeGuru over a code and validate that two high vulnerabilities are detected, which may be observed in Figure 2a.
2) Resolve the previously identified vulnerabilities.
3) Introduces a critical vulnerability related to SQL injection.
4) Execute CodeGuru over the code and validate if the implanted vulnerability is detected, which may be observed in Figure 2b.

Upon reevaluating the initial state, it is observed that while CodeGuru correctly identifies the remediation of the existing vulnerabilities, it fails to detect the new vulnerability introduced. Figure 3 illustrates that ChaosXploit concludes the experiment in a critical state, i.e., a rebuttal of the hypothesis. In other words, CodeGuru fails to identify security issues when confronted with a new common vulnerability.

Is chaos the future trend or just another keyword?

When applying the SCE methodology, a frequent question may arise: does SCE serve as a substitute for the traditional security testing approach? From our analysis, the response is negative. Nevertheless, introducing chaos enables testing particular risk scenarios that might not align with an automated security test or could be overlooked by an ethical hacker due to insufficient comprehension of the business context. That is, SCE does not replace traditional security tests but complements them by defining and validating specific hypotheses about the proper function of defense mechanisms. Through SCE, it is possible to introduce controlled chaos in a system, e.g., injecting small bugs in a system security control or even suppressing a control, so the application security team may analyze how the system and other existing security controls behave, enabling a zero-trust approach.

Additionally, efforts behind the application of SCE methodology are concentrated on realistic scenarios that consider the most common paths that would be followed by an attacker and that can not be tested using SAST or DAST tools. In the presented use case, different SCE experiments emerge aiming to validate the functionality of specific detective and corrective security controls that could interrupt the execution of real attacks. Such a process is possible since the proposed SCE experiments leverage an Attack-Defense Tree that orders the attack procedures by the attacker's purpose and effort.

Even if traditional security testing procedures and SCE can complement each other, they also have fundamental differences in many aspects. First, traditional testing is almost always executed by external personnel of the organization, while SCE is generally executed by internal personnel who know the steady state of the system and may propose appropriate tests.

Furthermore, the methodological approach also changes. While traditional security testing is bounded by ethical hacking methodologies such as OWASP, ISECOM, or EC-Council, SCE implements an ap-

proach addressed by the scientific method. As seen in the presented use case, the scientific method behind SCE is developed through some well-defined steps around the definition of observability's metrics, the proposition of a hypothesis according to the steady state, execution of an attack (injection of controlled chaos), and validation/rejection of the hypothesis.

On the other hand, the nature of vulnerabilities to be identified also experiences a crucial shift. In this sense, traditional security testing concentrates on pinpointing flaws in software operating in pre/production environments. With SCE, however, it becomes feasible to assess security at every phase of the software life cycle. As illustrated in the use case, SCE experiments can be devised for stages such as design, development, building, or deployment, integral components of a full-fledged DevSecOps practice.

Moreover, notable distinctions exist in the extent of automation, given that SCE attempts to automate all SCE experiments while traditional security testing still relies on numerous manual procedures. Consequently, having a dedicated SCE tool, such as ChaosXploit, becomes imperative. Such tools can seamlessly integrate into CI/CD pipelines, enabling the automation of attack procedures and supplying evidence to validate/refute a hypothesis.

Concerning the execution frequency, traditional security tests are typically conducted every 3 or 6 months, depending on the organizational policies. In contrast, SCE is designed to be executed regularly, aligning with the principles of DevSecOps, where software must be developed with both agility and security, ensuring an efficient and secure continuous deployment. SCE enforces security in a DevSecOps practice because the mode of operation of automated tools consists of a set of predefined rules for detecting vulnerability patterns, and the rigor of the rules is tied to the tool manufacturer.

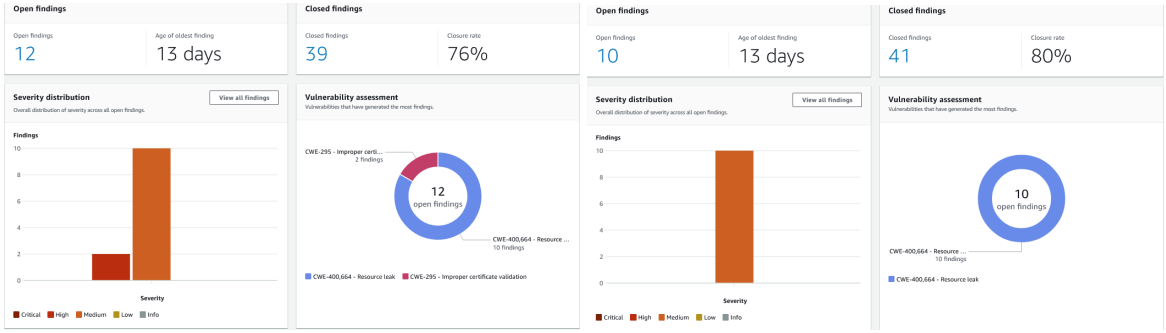
Finally, as seen in the use case, the SCE experiments integrate a defensive and offensive perspective through the Attack-Defense Tree, which integrates security controls and attack procedures in the same space. Such a dual perspective is highly appreciated for building better systems where countermeasures' implementation is motivated by specific security objectives. Traditional security testing, on the other hand, often relies on an only-offensive security approach.

ACKNOWLEDGMENTS

This work has been supported by Universidad del Rosario (Bogotá) through a "Beca de Estancia de Docencia e Investigación - EDI 2022-1", and by the Spanish Ministry of Universities linked to the European Union through the NextGenerationEU program, under Margarita Salas postdoctoral fellowship (172/MSJD/22).

REFERENCES

1. A. N. Novak and M. O. Vilceanu, ""the internet is not pleased": twitter and the 2017 equifax data breach," *The Communication Review*, vol. 22, no. 3, pp. 196–221, 2019. [Online]. Available: https://doi.org/10.1080/10714421.2019.1651595
2. K. Shortridge, "From lemons to peaches: Improving security roi through security chaos engineering," in *2022 IEEE Secure Development Conference (SecDev)*, 2022, pp. 59–60.
3. G. Blokdyk, *DevSecOps Strategy a Complete Guide - 2020 Edition*. Emereo Pty Limited, 2019.
4. A. Basiri, N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds, and C. Rosenthal, "Chaos engineering," *IEEE Software*, vol. 33, no. 3, pp. 35–41, 2016.



(a) CodeGuru findings before the implantation of code (b) CodeGuru findings after the implantation of code

FIGURE 2: CodeGuru findings during the SCE experiment

```
[2023-12-20 17:49:10 INFO] Steady state hypothesis: When a developer implements a SQL injection
 into application's source code AWS GuardDuty- Security identifies the issue.
[2023-12-20 17:49:10 INFO] Probe: Do I have Critical or high vulnerabilities?
Starting code guru session
Creating Session
Last scan created at: 2023-12-20 17:46:01.769000-05:00
{'critical': 0.0, 'high': 0.0, 'info': 0.0, 'low': 0.0, 'medium': 10.0}
Didn't find any critical or high issues
[2023-12-20 17:49:14 CRITICAL] Steady state probe 'Do I have Critical or high vulnerabilities?'
 is not in the given tolerance so failing this experiment
```

FIGURE 3: Output of the experiment # 1 executed in ChaosXploit

5. A. Rinehart, K. Shortridge, and a. O. M. C. Safari, *Security Chaos Engineering*. O'Reilly Media, Incorporated, 2020.
6. K. Shortridge and A. Rinehart, *Security Chaos Engineering: Sustaining Resilience in Software and Systems*. O'Reilly Media, 2023.
7. C. Rosenthal and N. Jones, *Chaos Engineering: System Resiliency in Practice*. O'Reilly Media, 2020.
8. K. A. Torkura, M. I. 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.
9. "Prowler pro," https://prowler.pro/, accessed: 2023-10-30.
10. S. Palacios, P. Nespoli, D. Díaz-López, and Y. Niño Roa, "On the way to automatic exploitation of vulnerabilities and validation of systems security through security chaos engineering," *Big Data and Cognitive Computing*, vol. 7, no. 1, 2023. [Online]. Available: https://www.mdpi.com/2504-2289/7/1/1
11. C. Polop, "Cloud HackTricks." [Online]. Available: https://cloud.hacktricks.xyz/pentesting-cloud/aws-security

Martín Bedoya is leader of the Hacking Center at NTT Data, Bogotá, Colombia. He is a computer engineer with experience in software development, ethical hacking and security in the software life cycle. He is also student of the MSc in Applied Mathematics and Computer Science at the University of Rosario (Colombia). Contact him at martin.bedoya@urosario.edu.co.

Sara Palacios is an Operational Specialist (Web Application Security) at Baufest. She is a professional in Applied Mathematics and Computer Science at the University of Rosario (Colombia) and current student of the MSc in Systems and Computing Engineering at Los Andes University (Colombia). She helps different software development teams to improve their secure coding practices and collaborates with the definition and implementation of security policies, standards and procedures. Contact her at sara.palaciosc@urosario.edu.co.

Daniel Díaz-López is a PhD in Computer Science from University of Murcia (Spain). He is professor for the School of Engineering, Science and Technology at the University of Rosario, Colombia and visiting professor for the Tandon School of Engineering at New York University, USA. His research lines are: cyber intelligence, privacy-preserving mechanisms, secure software development lifecycle, ethical hacking and security for Internet of Things (IoT). Contact him at danielo.diaz@urosario.edu.co.

Pantaleone Nespoli, is a postdoctoral researcher working together with the Department of Information and Communication Engineering at the University of Murcia, Spain, and the SCN team of the SAMOVAR laboratory, at Institut Polytechnique de Paris. His research is focused on cybersecurity and cyber defense training, with a particular interest in the detection and response to intrusions, and disinformation in social networks. Contact him at pantaleone.nespoli@um.es.

0.5 Conclusions and Future Works

- When a company implements an Application Security Program, it is essential to define security enablers to guarantee the effectiveness of the program. Thus, LLMs allow automating security enablers in all steps of the development lifecycle, as demonstrated in this degree work. Such automatization is translated into security task acceleration in the software lifecycle. In this regard, automating one of the most important enablers, i.e. threat modeling, reduces the time spent by the software factory to identify the necessary countermeasures to deploy secure systems in production.

- A company can increase the maturity level of its application security program by using Security Chaos Engineering because through this practice it is possible to identify vulnerabilities that are not detectable with automatic tools or pen-testing. This is due to in a SCE methodology, the system is challenged to eventual security flaws and the behavior of its components and stakeholders is analyzed. As demonstrated in this master thesis, over-reliance on security tools can be a risk factor when these tools fail.

- It is possible to apply Security Chaos Engineering to a DevSecOps practice from 2 perspectives. The first one (manual) introduces small failures in the components of a DevOps infrastructure to identify risk vectors in the development ecosystem, and the second one (automatic) uses the continuous integration or continuous deployment pipelines to launch experiments in an automated way, for example by using tools such as ChaosXploit, which can be integrated to the pipelines.

- As future work we plan to test the effectiveness of other LLMs, such as Bard [15], Bloom [16] or GPT-5 [17] to create a threat model based on attack-defense trees using the flowchart proposed in Section 0.4.3 and explore other threat modeling methodologies that can be automated using LLM's, for example, STRIDE [18], PASTA [19], TRIKE [20], DREAD [21], among others.

- On the other hand, it is also planned to create a flowchart with a series of prompts that allows the creation of attack-defense trees using LLMs but is oriented to identify cases of abuse in specific functionalities of a system. A case of abuse could be associated with malformations in the business logic, for example, bugs in the provisioning of accounts, bugs in monetary transactions, bugs in the privileges granted to an object, among others, and testing them in an automated way using Security Chaos Engineering.

Bibliography

[1] United States Code. Sarbanes-oxley act of 2002, pl 107-204, 116 stat 745. Codified in Sections 11, 15, 18, 28, and 29 USC, July 2002.

[2] Centers for Medicare & Medicaid Services. The Health Insurance Portability and Accountability Act of 1996 (HIPAA). Online at http://www.cms.hhs.gov/hipaa/, 1996.

[3] Edward A. Morse and Vasant Raval. Pci dss: Payment card industry data security standards in context. *Computer Law & Security Review*, 24(6):540–554, 2008.

[4] OWASP Foundation. The owasp application security program quick start guide. Accessed: 12-01-2024.

[5] OWASP Foundation. Opensamm. Online at https://www.opensamm.org/, Mar 2011. Accessed: 12-01-2024.

[6] Herb Krasner. The cost of poor software quality in the us: A 2020 report. *Proc. Consortium Inf. Softw. QualityTM (CISQTM)*, pages 1–46, 2021.

[7] OWASP Foundation. The zap homepage. Online at https://www.zaproxy.org/. Accessed: 19-01-2024.

[8] PortSwigger. Burpsuite - application security testing software. Online at https://portswigger.net/burp. Accessed: 19-01-2024.

[9] Snyk. Developer security: Develop fast. stay secure. Online at https://snyk.io/. Accessed: 19-01-2024.

[10] Opentext. Fortify application security. Online at https://www.microfocus.com/es-es/cyberres/application-security. Accessed: 19-01-2024.

[11] 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.

[12] OWASP Foundation. Owasp top ten. Online at https://owasp.org/www-project-top-ten/. Accessed: 23-01-2024.

[13] Mark E. Haase. Top 25 software errors — sans institute. Online at https://www.sans.org/top25-software-errors/. Accessed: 23-01-2024.

[14] Guntur Budi Herwanto, Gerald Quirchmayr, and A Min Tjoa. A named entity recognition based approach for privacy requirements engineering. In *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, pages 406–411. IEEE, 2021.

[15] Google. Google bard: A conversational ai tool by google. Online at https://bard.google.com/. Accessed: 28-01-2024.

[16] Huggingface. Bigscience large open-science open-access multilingual language model. Online at https://huggingface.co/bigscience/bloom. Accessed: 28-01-2024.

[17] Jose Antonio Lanz. Gpt-5 is officially on the openai roadmap despite prior hesitation. Online at https://decrypt.co/206044/gpt-5-openai-development-roadmap-gpt-4, Nov 2023. Accessed: 28-01-2024.

[18] Shawn Hernan, Scott Lambert, Tomasz Ostwald, and Adam Shostack. Uncover security design flaws using the STRIDE approach. *MSDN Magazine*, Nov. 2006.

[19] Tony UcedaVelez and Marco M Morana. *Risk Centric Threat Modeling: process for attack simulation and threat analysis*. John Wiley & Sons, 2015.

[20] Paul Saitta, Brenda Larcom, and Michael Eddington. Trike v. 1 methodology document [draft]. *URL: http://dymaxion. org/trike/Trike v1 Methodology Documentdraft. pdf*, 2005.

[21] Archana Singhal, Hema Banati, et al. Fuzzy logic approach for threat prioritization in agile security framework using dread model. *arXiv preprint arXiv:1312.6836*, 2013.