big data and cognitive computing

MDPI

Article

On the Way to Automatic Exploitation of Vulnerabilities and Validation of Systems Security through Security Chaos Engineering

Sara Palacios Chavarro [1], Pantaleone Nespoli [2], Daniel Díaz-López [1,3,*] and Yury Niño Roa [4]

[1] School of Engineering, Science and Technology, Universidad del Rosario, Bogotá 111321, D.C., Colombia
[2] Department of Information and Communications Engineering, University of Murcia, 30100 Murcia, Spain
[3] Tandon School of Engineering, New York University, Brooklyn, NY 11201, USA
[4] Cloud Infrastructure Engineering, Google, Bogotá 111321, D.C., Colombia
* Correspondence: danielo.diaz@urosario.edu.co or daniel.diaz@nyu.edu; Tel.: +57-2970200

Abstract: Software is behind the technological solutions that deliver many services to our society, which means that software security should not be considered a desirable feature anymore but more of a necessity. Protection of software is an endless labor that includes the improvement of security controls but also the understanding of the sources that induce incidents, which in many cases are due to bad implementation or assumptions of controls. As traditional methods may not be efficient in detecting those security assumptions, novel alternatives must be attempted. In this sense, Security Chaos Engineering (SCE) becomes an innovative methodology based on the definition of a steady state, a hypothesis, experiments, and metrics, which allow to identify failing components and ultimately protect assets under cyber risk scenarios. As an extension of a previous work , this paper presents ChaosXploit, 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. The use of ChaosXploit may be part of a defensive security strategy to detect and correct software misconfigurations at an early stage. Finally, different experiments are described and executed to validate the feasibility of ChaosXploit in terms of auditing the security of cloud-managed services, i.e., Amazon buckets, which may be prone to misconfigurations and, consequently, targeted by potential cyberattacks.

Keywords: security chaos engineering; attack trees; cloud managed services; vulnerabilities

check for updates

Citation: Palacios Chavarro, S.; Nespoli, P.; Díaz-López, D.; Niño Roa, Y. On the Way to Automatic Exploitation of Vulnerabilities and Validation of Systems Security through Security Chaos Engineering. *Big Data Cogn. Comput.* **2023**, *7*, 1. https://doi.org/10.3390/bdcc7010001

Academic Editors: Peter R.J. Trim and Yang-Im Lee

Received: 17 October 2022
Revised: 16 November 2022
Accepted: 22 November 2022
Published: 20 December 2022

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Protecting Information and Communication Technology (ICT) assets against potential threats is nowadays essential, especially with the advent of industry 4.0 and the consequent revolution. To this extent, cybersecurity aims to protect data and technological infrastructure in different spheres, e.g., personal, familiar, business, and social. In fact, different efforts have been made to contribute in such ways, for example, to protect persons against online sex offenders [1], to defend IoT devices from attacks against data or services [2], to make smart cities' infrastructure more resilient [3], to implement cybersecurity in distributed organizations [4], and to support LEA's (Law Enforcement Agencies) in the detection of malware [5] or in the prevention of cybercrimes [6]. Additionally, cybersecurity has also been considered a field of knowledge that goes beyond the validation of identity, protection of access, and monitorization of actions. Indeed, it has become a field that focuses its efforts on the consistency and resilience of systems.

Besides, Site Reliability Engineering (SRE) is a set of practices that aims to improve a system's design parameters and the conditions where it operates to supply the system with essential attributes such as scalability, reliability, and efficiency. The SRE concept originated at Google around 2003 and was rapidly adopted by other companies with strict

software requirements regarding scalability and reliability [7]. SRE may be seen as one way to materialize a DevOps strategy as it offers a set of principles around automatization, quantification of business-required reliability, reduction of availability risks, and observability. SRE may be implemented through the definition of reliability goals such as SLI (Service Level Indicator) or SLO (Service Level Objective), the development of a capacity plan, and the definition and execution of a change management process, among others [8].

A relatively new approach in the scope of SRE used to test the resiliency of distributed systems has recently emerged, known as Chaos Engineering (CE). CE is used to validate a system's strengths and vulnerabilities when exposed to uncontrolled conditions. By leveraging the CE methodology, different tests may be designed and applied with the aim of validating in a measurable way the changes that the steady state of a system may experiment [9]. Furthermore, another CE principle refers to the importance of including real word events (hardware or software failures) in the experiments, especially events that have the potential to generate a high impact or may occur with some frequency. CE also remarks on the importance of automating experiments as much as possible as it allows for a better analysis of the outcomes. Lastly, CE prioritizes the execution of tests in production to guarantee authenticity in the experiments and to consider real traffic patterns, although the impact of such experiments should be carefully estimated and contained.

Following the CE methodology, a "chaotic" experiment must be designed over a controlled environment, which allows the observation of the variables that define the steady state of the target system. Additionally, such a CE experiment must be ruled by a scientific method that allows the definition and validation of a set of hypotheses [10]. Lately, CE experiments have gained importance as a way to implement SRE as it allows testing the resiliency of a system against chaotic events so that the system's weaknesses can be identified and corrected in advance. Nonetheless, the resiliency of a system should be validated not only from a perspective of availability. In fact, it should include other aspects related to the secure and correct operation of the system. Thus, the necessity of evaluating the system's resiliency in a holistic way emerges and is consistently most demanded when we evaluate distributed systems that manage sensitive information, such as secure IoT services [11] or personal data management solutions [12].

Intending to execute a security-based evaluation of a system, a fresh concept emerged in 2017 to apply CE principles to experiments that, together with the availability, evaluate the confidentiality and integrity of a system under chaotic events. That is, SCE (Security Chaos Engineering) joins the cybersecurity ecosystem, trying to defend the systems against such events. In a cybersecurity context, chaotic events may be generated by a threat agent that tries to: (i) make a system unavailable, e.g., through a Distributed Denial of Service (DDoS) attack, (ii) read sensible data hosted by a system, e.g., through an elevation of privileges that facilitate the access to restricted information, or (iii) modify users or system files that alter the operation of the system, e.g., through the remote execution of malicious code [13].

Noting that the CE methodology can significantly impact new developments by reducing vulnerabilities through the scientific method and experimentation, this paper addresses the following research question: how can SCE be used to detect application vulnerabilities automatically, not limited to a specific context and by taking into account the actions that are preferred by an attacker based on the effort expended in the exploitation?

Thus, the current paper proposes an SCE framework based on attack trees named ChaosXploit. ChaosXploit is expected to support the operations of security teams in charge of detecting and correcting in an anticipated way the vulnerabilities that an under-analysis system may contain. The defensive labor of those teams implies understanding the attack goal that an attacker may pursue as well as the offensive techniques that he/she may use to achieve such an attack goal.

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

1. The proposal of ChaosXploit, an SCE framework that leverages attack trees to address the execution of attacks. Particularly, the ChaosXploilt architecture contains three main components: an observer, an experiment runner, and a knowledge database;
2. The design of an attack tree that pursues a specific attack goal, i.e., the extraction or modification of AWS S3 buckets information that enriches the knowledge database of ChaosXploit;
3. The execution of a set of experiments that validates the feasibility of ChaosXploit to execute an attack tree over a specific target, i.e., AWS S3 bucket, exposing multiple misconfigurations.

ChaosXploit was first presented in Ref. [14], and the present paper is an extended version of such work to include the following improvements and new content:

1. An extension of Section 2 (State of the art) with a detailed analysis of the related works, including a comparative table with six identified key features;
2. The addition of Section 3 (Background), which includes a set of essential concepts that introduces the reader to SCE;
3. An extension of Section 5 (Experiments) resulting in the implementation and execution of the second branch of the proposed attack tree, which aims to extract or modify information from the AWS S3 buckets;
4. The addition of Section 6 (Discussion), which includes an analysis of the current and future adoption of SCE in the industry.

The remainder of this paper is organized as follows: Section 2 gathers the major works contributing to SCE, analyzing their strengths and weaknesses. Then, Section 3 explains the fundamentals and concepts regarding CE and SCE. In Section 4, ChaosXploit, our proposed framework to execute SCE experiments, is described. In Section 5, diverse experiments to test ChaosXploit are designed and performed. Section 6 presents an engaging discussion on the adoption of SCE in enterprises. Finally, Section 7 concludes the work, adding future work to possibly improve ChaosXploit.

2. State of the Art

Throughout the literature, CE appears as a relatively hot research topic. That is, its robust capabilities have been described in different research items while being applied in several contexts. Nonetheless, such an application and a proper definition must be clarified since they have been ambiguous so far.

Starting with Netflix's release of *Chaos Monkey* in 2011 [15], the CE paradigm has been mainly used to test the resilience and robustness of virtualized appliances, demonstrating the potentialities of the chaotic method in such scenarios.

To this extent, the work in Ref. [16] described *Pystol*, a fault injection framework to argue on the resiliency of hybrid-cloud systems in adverse events. Specifically, *Pystol* is presented as a Software Product Line (SPL) that can be mounted on top of cloud infrastructures, being able to exploit CE's capacities. The proposal is then developed in a production environment and deployed using standard Kubernetes objects (together with the corresponding APIs) and Amazon Web Services (AWS) to execute the entire cluster with three use cases. It is worth mentioning that *Pystol* has been made available as an open-source code for further community development.

Additionally, Simonsson et al. [17] proposed *ChaosOrca*, another open-source fault injection platform for system calls in containerized applications based on CE principles. In this sense, *ChaosOrca* can calculate the self-protection ability of Docker-based microservices with regard to system call errors. In particular, the system determines the steady state of the Docker container by systematically registering diverse system metrics (CPU and RAM consumption, network I/O, among others). Later, some perturbations are injected into the system calls executed by the isolated dockerized app, avoiding the possible impact on the ordinary operations of other containers. The proposal is tested in three Docker microservices scenarios, namely Torrent, Bookinfo, and Nginx, demonstrating encouraging results in noticing resilience flaws.

An interesting case study on applying the CE methodology to a real use-case scenario has been conducted in Ref. [18]. The main idea of the authors is to introduce the CE paradigm at ICE Gruppen AB, a group of companies working in the grocery market. Mainly, they started with a literature review, studying the state-of-the-art works on CE and performing explanatory interviews in the company. The resulting framework, based on a total of 27 open source CE tools, is then applied to the IT system of the company, including its e-commerce. Interestingly, among the CE categories identified during the process, the authors also indicate "network attacks" and "security attacks".

Furthermore, *ChaosMachine* is described by Zhang et al. [19]. Particularly, it can be defined as an open-source and extensible CE framework written in Java to analyze the capacities of handling exceptions in production environments. In this sense, *ChaosMachine* is able to disclose possible resilience issues of try-catch blocks, proposing an architecture composed of three parts: (i) a monitoring sidecar, (ii) a perturbation injector, and (iii) the chaos component. Then, *ChaosMachine* is tested with three voluminous open-source Java apps, totaling 630k code lines, exhibiting its capacities in production environments with realistic workloads.

Lately, the principal objective of the chaotic methodology has changed, shifting from resilience surrounding a system to enclosing security issues. Starting from the assumption that security failures will happen doubtless, SCE's primary goal is to test the system's security controls using proactive experiments and, therefore, building confidence in its capabilities to protect against potential threats.

Lamentably, since this paradigm shift has recently happened, the quantity of academic items and tools is still insufficient. In this sense, *ChaoSlingr* can be depicted as the first open-source software contribution to exhibit the potential application of the chaotic principles to information security [20]. The tool was developed to function on AWS by a team at the UnitedHealth Group, led by Aaron Rinehart, to demonstrate a simplified mode for designing security chaos experiments [21]. From the main project, several companies have started to leverage *ChaoSlingr* to execute chaotic experiments within their systems.

Moreover, Torkura et al. [13] proposed *CloudStrike*, a software architecture that measures the security of cloud environments by applying Risk-Driven Fault Injection (RDFI). For the reader's sake, the tool was first proposed in a previous article [22]. Concretely, RDFI expands the CE paradigm to contemplate cloud security without losing the resilience perspective by injecting security faults, leveraging the attack graphs representation. Such SCE tool is tested on various cloud services of principal platforms, namely, AWS, and Google Cloud Platform. Notably, the authors claim that they can calculate the risk value to which the system's assets are being exposed to by using the Common Vulnerability Scoring System (CVSS). Then, the authors used the SCE methodology to test another tool, *CSBAuditor*, a cloud security framework that can continuously monitor cloud infrastructures to identify possible ill-motivated activities [23].

Additionally, the application of SCE to enhance API security is defined in Ref. [24]. Due to the popularity of RESTful APIs in distributed applications, the authors propose utilizing this methodology to test the configuration of the API's security controls, exposing early vulnerabilities. After focusing on the OWASP (Open Web Application Security Project) list of the top 10 critical web application security risks and automated attack detection, the authors suggest the application of SCE experiments to address the abovementioned challenges. Indeed, the work is still in an early phase, but the capabilities of SCE are recognized as being valuable.

Besides, SCE experiments have been used to test System of Systems (SoS) robustness against potential attackers in [25]. Concretely, the authors used Chaos Toolkit to conduct several CE and SCE experiments on a Virtual Unmanned Aerial Vehicle (VUAV). The Attack Trees methodology is employed to better model possible attacker moves, assuming the level of access he/she would possess with a previous threat modeling phase. Precisely, two Attack Trees are developed, namely, injecting corrupted navigation service and killing ActiveMQ/WorldWind (i.e., the software tools used for communication purposes). Then,

five separate experiments are executed, evaluating the performance by measuring the CPU and RAM usage. Results showed a slight increase in CPU load, while RAM was not a significant metric during tests.

Table 1 summarizes the findings of the state-of-the-art investigation. It has to be remarked that the works [16–19] refer to CE applications while [13,24,25] propose SCE employment. Consequently, one could argue that it is obvious that the attributes' value of the CE works tend to be "Resiliency", while "Security" is predominant for the SCE proposals. Nevertheless, the proposed framework in Ref. [18] adds security features to the CE requirements. Such confusion is directly derived from the ambiguous definition of CE, as previously stated.

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

Related Work	Attributes	Application Context	CE Tool	Threat Model	Experimental Data	Automation Level
Camacho et al. [16]	Resiliency	Cloud	Ad-hoc	✗	✗	✗
Simonsson et al. [17]	Resiliency	Containers (Docker)	Ad-hoc	✗	crafted	✓
Jernberg et al. [18]	Resiliency Security	Web	27 CE tools survey	✗	crafted	✗
Zhang et al. [19]	Resiliency	Java applications	Ad-hoc	✗	Public Java code	✓
Torkura et al. [13]	Security	Cloud	Ad-hoc	Could Attack Graphs	crafted	≈
Sharieh, Ferwron [24]	Security	API	✗	✗	✗	✗
Bailey et al. [25]	Security	SoS	ChaosToolkit	Attack Trees	crafted	✗
Our proposal ChaosXploit	Security	Any	ChaosToolkit based	Attack Trees	Public buckets	✓

Legend: ✓ Yes, ✗ No, ≈ Partially.

Another clear difference between CE and SCE works is that most SCE proposals leverage a threat model to map the attackers' moves within the protected system. In particular, Attack Graphs and Attack Trees seem to be a suitable choice to infer the goals of the attackers and, possibly, anticipate them.

Regarding the tool used to implement the proposals, many of them present ad hoc development of the CE/SCE framework. In this sense, one could say that, in specific situations, implementing from scratch can lead to better solutions. However, re-using already mature and tested tools should be the primary choice in order to fairly compare different proposals.

Additionally, two key aspects must be highlighted: (i) the importance of using publicly available data to perform experiments and (ii) the significance of a high automation level for CE/SCE frameworks. That is, most of the analyzed papers present crafted experiments to demonstrate their features, making the comparison challenging to execute. Then, one of the crucial characteristics of any chaotic tool is the automation level of the experiments. Since modern systems feature high complexity and distribution, automating those experiments is highly desirable.

Last but not least, the surveyed works suggest the chaos tests application only in a particular context (e.g., Cloud, containers, etc.). It is effortless to claim that the design of a full-fledged CE/SCE tool would broaden its application scope, leading to more experimentation and, perhaps, better results.

The research presented in the paper at hand uses as reference the characteristics of all these tools presented in related works and proposes an SCE-powered framework based on attack trees to detect and exploit vulnerabilities in different targets as part of an offensive security exercise. This framework, unlike those previously mentioned, can be used in any application context whether in different clouds (AWS, GCP, Azure), containers (Docker, Kubernetes), or web applications. Additionally, compared to current CE tools, our proposal develops a threat model based on attack trees since these enable modeling organized actions for more than one SCE experiment, allowing a better traceability and following the same attack goal. Another differentiating component that stands out in our proposal vs. other SCE tools is the high level of automation, since we can make a list of actions to be performed and, when launching the experiment, these will be executed in a row. Finally, we are aware that we are not reinventing the wheel, as our proposal is built on

the ChaosToolKit, one of the most mature tools in CE. Lastly, our proposal is tested with common cloud services, meaning that the experiments can be easily replicated.

3. Background

For the sake of the reader, some important concepts are introduced to allow a better understanding of the context surrounding CE and SCE.

3.1. Chaos Engineering (CE)

As previously mentioned, the concept of CE emerged in 2011 when Netflix moved its services to the AWS cloud. Netflix's engineers feared that an internal instance could fail during the move, severely impacting the overall operation. For this reason, *ChaosMonkey* was created to test the stability of Netflix by injecting faults that randomly terminate internal instances [26]. A year after launching *ChaosMonkey*, Netflix added new modes that report different types of faults or detect abnormal conditions. Each of those modes were considered to be a new simian, and together they formed what is known as the *SimianArmy* [27].

In 2016, Kolton Andrus and Matthew Fornaciari founded Gremlin [28], which is recognized as a leading CE solution. Along with the creation of Gremlin, the formal definition of CE was also born as "the discipline of experimenting on a system in order to build confidence in the system's capability to withstand turbulent conditions in production" [9].

A few CE frameworks may be found in the wild. One of the most notable frameworks is the above-mentioned Gremlin, which allows one to experiment with more than 10 different attack strategies on different infrastructures. Nevertheless, not all of those strategies are free to use, and it does not have reporting capabilities. Another well-known CE framework is ChaosMesh [29], an open-source cloud-native tool built on Kubernetes Custom Resource Definition (CRD). Specifically, it allows testing several scenarios checking for network latency, system time manipulation, and resource utilization, among others. Nonetheless, this tool does not have the advantage of scheduling attacks.

Another open-source CE framework is Litmus [30] which allows developers to use a set of tools to create, facilitate, and analyze chaos in Kubernetes with automatic error detection and resilience scoring. Last but not least, it is important to mention ChaosToolkit (CTK) [31], an open-source tool that permits the automation and customization of CE experiments by defining a set of probes and actions that may be pointed to different types of targets.

It is worth remarking that the CE experiments are not chaotic at all. In fact, they are based on the scientific method and should follow the CE principles [9] that define the subsequent steps to guarantee that the experiments are correctly executed.

1. Define the behavior of the system (**observability**), which is key to measure with the purpose of approval or disapproval of an hypothesis that may be defined later;
2. Identify the **steady state** to mark out what should be considered as a normal behavior of the system;
3. Define a **hypothesis** that will be proved or refuted at the end of the experiment;
4. **Execute** the experiment by introducing real-world events such as creating instances that expose malfunctions and interrupted network connections, among others.

The fact that CE experiments have a defined method corroborates that this discipline does not consist of "breaking things on purpose". On the contrary, CE experiments are generally done in a proper testing environment with similar conditions to the ones obtained in a real environment exposed to disruptive incidents. Thus, the application of CE allows testing attributes such as availability and reliability in a controlled environment. Generally, the results that arise from conducting a CE experiment can help anticipate incidents, improve the understanding of system failure modes and reduce maintenance costs [32].

Once the method and benefits of implementing CE have been discussed, defining and implementing an experiment can be effortless. For example, in Ref. [10], one of the experiments considered a recommendation system that, as part of its functionality, stores

all the searches inserted by users in a cache so that such queries may be used to redefine the recommended product that is returned to the user. The experiment uses CE to check what would happen if the communication were to fail between (i) the process (Redis Client) requesting to store the queries and (ii) the cache (Redis server) that effectively stores them. In this case, the purpose of the CE experiment is to determine if the recommendation system may still work after injecting failures, so it is defined as follows:

- **Observability**: Navigate the application and view the recommended products;
- **Steady State**: The recommended products should be returned to the user;
- **Hypothesis**: A failure in the communication with the storage component (Redis server) causes a failure in the product returned to the user by the recommendation system, even in subsequent queries when the storage component is restored.

With the **execution** of this experiment it may be possible to conclude, for example, that the hypothesis is refuted since when injecting failures in Redis Server, the recommendation system handles the error and manages to recover automatically as soon as the access to the storage system is re-established. Thus, it proves that the recommendation system is resilient to failures in the storage system.

3.2. Security Chaos Engineering

By using CE, testing security in systems with the premise that "failure is the greatest teacher" is possible. This idea was first proposed by Aaron Rinehart [21], who pursued the application of CE in cybersecurity while working as Chief Security Architecture at the UnitedHealth Group [33]. As mentioned in the previous section, CE has traditionally focused on testing system availability, while recent research is striving to apply this discipline in the field of cybersecurity. Concretely, the main goal is to apply CE concepts by testing not only the availability but also other attributes such as integrity and confidentiality to boost the concept of *Security Chaos Engineering* (SCE). SCE has been defined as "the identification of security control failures through proactive experimentation to build confidence in the system's ability to defend against malicious conditions in production" [21].

In this context, ChaosSlingr can be recognized as the first open-source framework that demonstrated the value of applying SCE to cybersecurity [34]. This tool was created by Aaron Rinehart and proposed a simple experiment. It sought to misconfigure some ports on a system and observe the behavior. Although it was a good initiative, ChaoSlinger was no longer maintained and became part of a larger project known as Verica [35].

As mentioned, while CE aims to test the resilience of a system, SCE also provides measures and experiments to provide top-notch security to the systems. By leveraging the SCE methodology, it is possible not only to corroborate assumptions or discover vulnerabilities but also to infer possible mitigations [36]. That is, SCE falls into the cybersecurity ecosystem, as it allows checking that the security controls that validate the confidentiality, integrity, and availability of the system are reliable. This check is based on identifying security flaws caused by the human component, insecure design, and lack of resilience in the system under protection. In addition, SCE experiments can identify the exact points where security flaws exist and act on time.

The methodology applied by SCE is similar to the one described for CE, as it incorporates the definition of steady state, observability, and hypothesis. However, it pursues a different objective as it aims to validate the security of a system, for example, by discovering vulnerabilities, misconfigurations, logic flaws, and insecure design, among others. In addition, if experiments are executed frequently, SCE may help in the reduction of security incidents and remediation costs, as it allows developers to: (i) understand their system, (ii) define a response plan, (iii) identify system modules failing, and (iv) note that some components were omitted during development. In addition, SCE minimizes impacts on users through experimentation, which in turn improves the ability of developers to track and measure security.

One helpful experiment to explain the SCE methodology is associated with understanding the behavior of a firewall when some associated ports are misconfigured. This

was one of the experiments that were executed with *ChaoSlingr*, a framework created by a team at UnitedHealthGroup, explained in detail in Chapter 7: the journey to SCE of [21]. A brief overview of the experiment is presented below:

- **Observability**: Detection of security configuration changes that have occurred in a device;
- **Steady State**: The firewall is able to detect all changes over the ports;
- **Hypothesis**: A misconfigured port should be detected and blocked by the firewall, and such an event should be appropriately logged.

From the **execution** of this experiment, it could be possible to prove that half of the time, the hypothesis is fulfilled, and the other half of the time, the firewall does not detect and block it. In addition, a cloud configuration tool could be able to detect the failure, but this is not being logged, so it is not possible to identify that an incident has occurred. Thus, proper remediations should be undertaken to avoid the incorrect operation of the firewall.

3.3. Differences between SCE and Traditional Pentesting

At this point, one could legitimately wonder about the difference between SCE and traditional penetration testing techniques and the added value of using SCE. In order to establish these differences, Table 2 illustrates some key aspects to be considered in this comparison, which are explained in the following paragraphs.

Table 2. Main differences between traditional pentesting and SCE.

	Traditional Pentesting	SCE
People implementing	Executed mainly by personnel external to the organization (external red team)	Executed mainly by organization's internal personnel (internal red or blue team)
Methodology behind	ISECOM, EC-Council, OWASP, others	Chaos Engineering principles
Security approach	Offensive	Defensive
Available tools	Bunch of offensive tools	Few SCE frameworks
Grade of automatization	Mainly manual procedures	Mainly automated procedures
Expected frequency	Depend on organization policies and risk appetite, generally every 3 or 6 months	High frequency for definition, can be applied for each incremental development
Phase of SDLC where applied	Generally in production phase	Along all the SDLC
Scope of tests	Generally unitary tests	Unitary and full system tests
Kind of vulnerabilities detected	Own-system errors, misconfigurations	Own-system errors, security assumptions about the systems

As indicated in Table 2 traditional pentesting allows attacking different targets by finding and exploiting vulnerabilities and misconfigurations. On the other hand, SCE allows us not only to test for system errors but also for security assumptions about the system, which includes component misconfiguration but also human errors, so we can affirm that SCE has a bigger scope in terms of vulnerabilities that can be detected.

In addition, the pentesting process may require a set of different activities, which can be automated in a defined way, e.g., fingerprinting, scanning, and brute forcing, but the exploitation phase will generally require highly manual activities through the construction of customized exploits and payloads. Secondly, SCE strives for a high automatization in the development of experiments, so they can be reproducible and repeatable.

Additionally, traditional pentesting is generally executed by an external red team, because generally, the aim is to emulate a double-blind scenario where an attacker does not know the internal details about the system that he is attacking, and the persons in charge of protecting the system do not know when the attack will be launched [37]. In this regard, SCE offers a different approach, as SCE experiments are intended to be executed by the persons who build (developers), maintain, and secure the system, who can be part or not

part of a blue team or an internal red team in case the organization has one; all of this is part of a defensive strategy.

The frequency of pentesting exercises may depend on external regulatory or internal requirements and organization risk appetite, resulting in pentesting tests developed regularly, e.g., every 3 or 6 months for the case of organizations with an intermediate maturity security level, and mainly over systems that are in the production phase. In the case of SCE, the experiments have a high frequency by definition, as SCE experiments may be designed and performed along the software development life cycle. This means it is possible to incorporate it in the early stages of development and reduce the remediation costs.

It is important to note that currently there are many tools available that can be used in different phases of pentesting, but there are not many SCE-based tools, as indicated in Section 2, so the contribution of a framework in this regard improves the traditional pentesting process as it offers an alternative way of detecting vulnerabilities in the protected assets, providing a new tactic that enriches the existing tool-set of blue and red teams. Additionally, when considering complex or distributed systems, SCE experiments help to understand the system as a whole, going beyond unit tests over specific components which is common in pentesting exercises.

Finally, methodologies behind pentesting refer to quite popular publications from ISECOM (OSSTMM methodology), EC-Council (hacking phases), or OWASP (security testing guides), among others. However, none of them are based on a scientific method, which SCE does by following the CE principles.

4. ChaosXploit Architecture

This section describes ChaosXploit, a SCE-powered framework composed of different modules that support the application of CE methodology (described in Section 3.1) to test security in different kinds of information systems. The architecture of the proposal is depicted in Figure 1. It is worth noting that a label has been assigned to each module to represent the step in the EC methodology that is executed in that module. Additionally, each internal module is described in the following sections. In particular, the Knowledge Database is described in Section 4.1, the Observer is detailed in Section 4.2, and the SCE Experiments Runner is explained in Section 4.3.

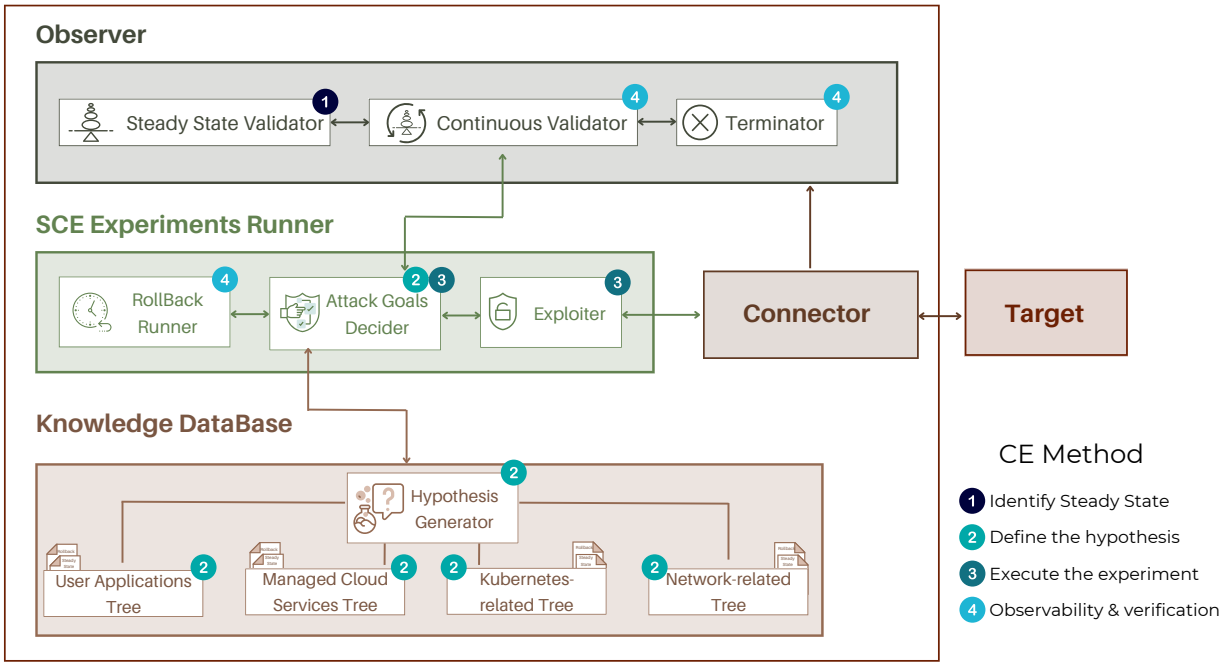

Figure 1. The proposed architecture of ChaosXploit and its relation to SCE methodology.

4.1. Knowledge Database

The knowledge database is responsible for providing the steps required to conduct an offensive SCE experiment executed by a team (blue team) interested in maturing a defensive strategy. Thus, this module is composed of a set of attack trees and a hypothesis generator, which will be some of those in charge of executing the second step of the CE methodology, i.e., defining the hypothesis for the experiment. The tasks assigned to these modules are detailed below.

4.1.1. Attack Trees

This module is in charge of delivering the intelligence for executing the SCE experiments. Such intelligence is represented by different attack trees, where each tree clusters different branches focused on achieving a specific attack goal, e.g., gaining access to data stored in a cloud storage solution. So, different attack goals may be pursued as attack trees are contained in the knowledge database. Each branch of an attack tree gathers different offensive actions that may be conducted to achieve the final attack goal, where an action may be a python script, an HTTP request, or some process to be run on the operating system. It is worth mentioning that attack trees for different types of targets may be defined, such as trees for user applications, managed cloud services, Kubernetes, and network devices, among others.

4.1.2. Hypothesis Generator

The intelligence contained in the attack trees needs to be converted to a hypothesis so that it can be consumed by the other modules of ChaosXploit. So, the Hypothesis Generator is responsible for translating the branch actions contained in an attack tree into a form readable for the module that executes the SCE experiments, i.e., the exploiter. Each hypothesis generated by this module is a statement about the system being tested that must be refuted or confirmed by the SCE experiments, e.g., an organization will not expose private data when the recognition tool Foca [38] is pointed out to the main domain.

4.2. Observer

The observer groups all the activities related to the observation of both the target and the SCE experiment. This module is important because it allows controlling the specific conditions of the target before, along, and after the execution of the SCE experiments.Therefore, this module will address, in its different components, the first step of the CE methodology: identification of the steady state and the fourth step: observability and verification. This module is composed of a steady-state validator, a continuous validator, and a terminator.

4.2.1. Steady State Validator

The steady-state validator is responsible for verifying the steady-state hypothesis on the target representing the steady-state conditions, which allows us to create a direct association with the first step of the CE methodology. These conditions will depend on the target of the attack and the hypothesis defined in the hypothesis generator. For example, a normal condition may be a well-formed response from a web server or an assumption about the system.

4.2.2. Continuous Validator

The continuous validator is activated once the experiment starts and is constantly checked until the end of the experiment. It allows for verifying specific signals detected from the target, which makes it possible to determine the results of an interaction between the exploiter and the target. These signals are especially important because they can indicate whether a current action included in a branch of an attack tree has succeeded, so the following action in the branch should be triggered, or they can indicate that the target is not vulnerable and the other actions in the branch should not be executed. This leads us

to categorize it as one of the components that perform the last step of the CE methodology, as it allows us to observe and verify the behavior of the experiment.

4.2.3. Terminator

Each time the execution of an action is completed, the experiment status is updated and the terminator validation is performed. This module observes the failure states of the SCE experiment to define the actions to be taken accordingly, thus it is associated with the last step of the CE methodology. For example, if the target stops responding due to the execution of an SCE experiment, the experiment status is updated to failed and the terminator will be able to inform the Rollback Runner so that it can restore the target.

4.3. SCE Experiments Runner

The SCE Experiments Runner is in charge of the SCE experiment's execution over a target to validate or refute a hypothesis. This component is fundamental because it not only leads the interaction with the target but also centralizes the communication with the observer and knowledge database. Although it is an execution module, it also includes elements that contribute to the development of the other steps of the CE methodology. It consists of three main elements: attack goal decider, exploiter, and rollback runner.

4.3.1. Attack Goal Decider

The attack goal decider receives a defined goal attack as input to be tested over a target. Such an attack goal may be contributed by the user of ChaosXploit who is interested in probing if a particular system is susceptible to a specific attack. Then, the attack goal decider requests the knowledge database for the proper attack tree that matches such a defined goal. This request implies that the module is involved in the hypothesis generation process (step 2 of the CE methodology). In addition, when asking for the information from the knowledge database, it will receive the actions to be performed to execute the experiment, which allows it to be associated with the third step of the methodology as well.

4.3.2. Exploiter

The exploiter executes the SCE experiment over a target to validate or refute a hypothesis. This is directly associated with the third step of the methodology. With such purpose, the exploiter performs the offensive actions defined previously by the attack tree obtained from the knowledge database. Besides, it is also able to collect information about specific responses coming from the target to define the next step in an attack.

4.3.3. Rollback Runner

An experiment may contain a sequence of actions that reverse what was undone during the execution; this allows us to identify the points where failures were generated. Thus, the Rollback Runner is supported by the last phase of the methodology. The set of actions will be called by the Rollback Runner after the Continuous Validator finishes its execution regardless of whether an error occurred in the process or not.

4.4. Connector

The connector is responsible for searching for the most suitable extension to connect to the target on which the user wants to run the experiment. Once an extension has been defined, the connector establishes the link with the target and tests that the scenario is adequate to run the SCE experiment.

While ChaosXploit has a high level of automation, some previous activities are required before executing the experiments. First, the security team in charge of testing an under-analysis system must define the attack goal to be tested in the experiments and draw an hypothesis with its corresponding steady state. Then, an attack tree consistent with the previously defined attack goal is needed, which may come from an external cyberthreat intelligence provider (in cases where the under-analysis system is common and sufficiently

known by the provider) or from the security team that builds it as a way to understand the possible steps an attacker could perform to achieve the attack goal. After the attack tree is defined, ChaosXploit will automatically perform all necessary actions, i.e., identify the vulnerability type, do the exploitation from the tree and measure steady-state, to conclude the SCE experiments. In case the results have not been satisfactorily completed, the type of vulnerability found will be indicated by ChaosXploit.

The interactions between the components of ChaosXploit are shown in Figure 2. First, the user of ChaosXploit requests the Attack Goal Decider for the execution of a SCE experiment, informing the attack goal to be considered and the target where the SCE experiment should be addressed. Then, the Attack Goal Decider retrieves from the knowledge database the steady-state of the experiment, the rollback procedure, and the most proper hypothesis (a branch in the attack tree) that matches the attack goal desired by the user. The Attack Goal Decider also requests to the Connector the preparation of the extension for the target informed by the user. When a connection to the target is established and a hypothesis is defined, the Attack Goal Decider then performs the following actions: (i) It establishes the steady state of the experiment in the Observer and tests it in an initial phase. Therefore, in this step, it is necessary to establish a new connection to validate its stability. In case this action fails, the state of the experiment is updated to failed and it is terminated; (ii) it starts the execution of the steps defined in the selected branch of the attack tree with the help of the Exploiter, and (iii) it keeps continuous communication with the Continuous Validator to monitor the execution of the exploitation in progress and in that way be aware if the attack goal is achieved. If the Continuous Validation fails, then the termination process is activated by the Terminator. The experiment ends with the execution of the Rollback Runner to restore everything.

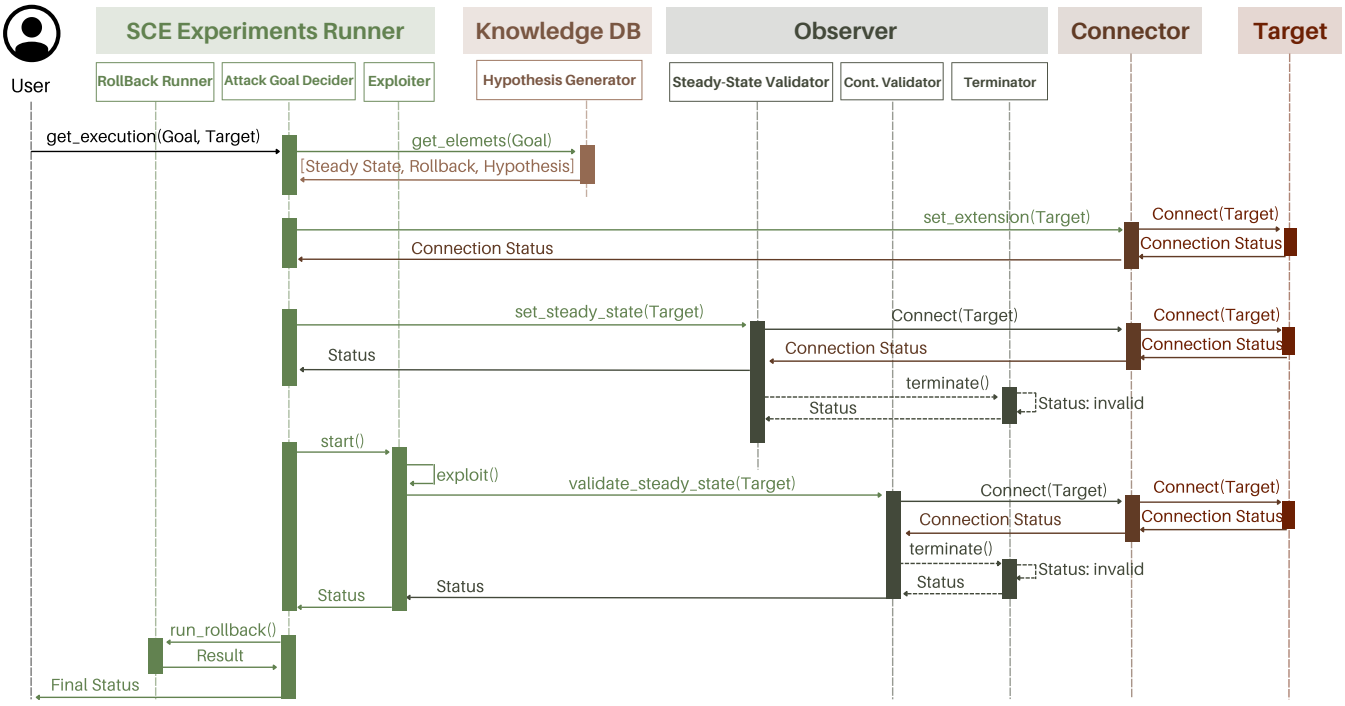

Figure 2. Flow diagram of the execution of a SCE experiment in ChaosXploit.

5. Experiments

Multiple experiments have been conducted using the ChaosXploit proposal mentioned in Section 4, which are also available in the public repository of the project [39]. Based on the fact that AWS S3 buckets and Elasticsearch databases account for nearly 45% of the cloud misconfigured and compromised technologies [40], the proposed session of ChaosXploit experiments focuses on evaluating the security of the AWS S3 service. It considers the

possible configurations and whether they permit establishing a connection, whether they are public or private buckets, or whether they permit getting the configured Access Control Lists (ACLs) which allow managing the access to the buckets and their objects. These lists define which AWS accounts or groups have access and what kind of permissions they have.

This section of experiments comprises the following subsections: Settings, Section 5.1, in which the hardware and software requirements to develop the experiment, are specified. Definition of the knowledge database, Section 5.2, in which the attack tree is presented together with the specification of the branches chosen for the experiments. Sections 5.3 and 5.4 describe the implementation of the first and second branches of the attack tree. Each of them contains the definition of the steady-state and the hypothesis, as well as the input parameters and the monitored variables. Additionally, each of them includes a subsection for result analysis.

5.1. Settings

The following setup was used to execute the above-mentioned experimental session using ChaosXploit:

- Hardware: the experiments were executed on a Fedora OS with AMD Ryzen 5 3500U CPU, 8 GB RAM, and 512 GB SSD;
- Internal Components: Some of the components of ChaosXploit have been built over existing modules of ChaosToolkit, as it is an open-source framework that allows its extension and improvement to make it oriented to security purposes. ChaosToolkit was chosen since this tool simply allows automation of the experiments using *json* files. The connection to the different targets (buckets) was done using boto3 (SDK for python);
- Environment: The first version of ChaosXploit should be installed on a virtual environment with *python3.7* and *Chaostoolkit* installed.

5.2. Definition of the Knowledge Database

In Figure 3 it is possible to observe the attack tree designed for this experimental session. In this case, ChaosXploit is used as an internal auditing tool where a user with the role of an attacker can follow the four paths shown in the attack tree. These paths are described as:

- **Branch 1**: In this case, the intruder first locates public buckets by either listing the names or by using search engines such as the Wayback Machine. The next step aims to verify whether the attacker is successful in connecting to the bucket. Once inside, he has access to look at the storage system's objects, and read the Access Control Lists (ACL). The attacker will be able to accomplish the attack objective if these ACLs have permissions that are available to the general public;
- **Branch 2**: In this route, the attacker tries to access private buckets using privilege escalation after failing to recognize public buckets. A policy rollback in this situation, where a user with permission to restore a previous policy is requested, presents a chance for privilege escalation. In a perfect world, this user would have had administrator rights or full access to the S3 service;
- **Branch 3**: in which the attacker can use brute-forcing techniques to compromise other user's credentials and thereby gain access;
- **Branch 4**: where the attacker can use social engineering techniques such as phishing to compromise credentials and gain access.

It is important to note that the execution of the first and second branches was included in the scope of this project, as the actions included in such branches were easier to automate. Other branches could also be implemented through a combination of manual and automatic actions.

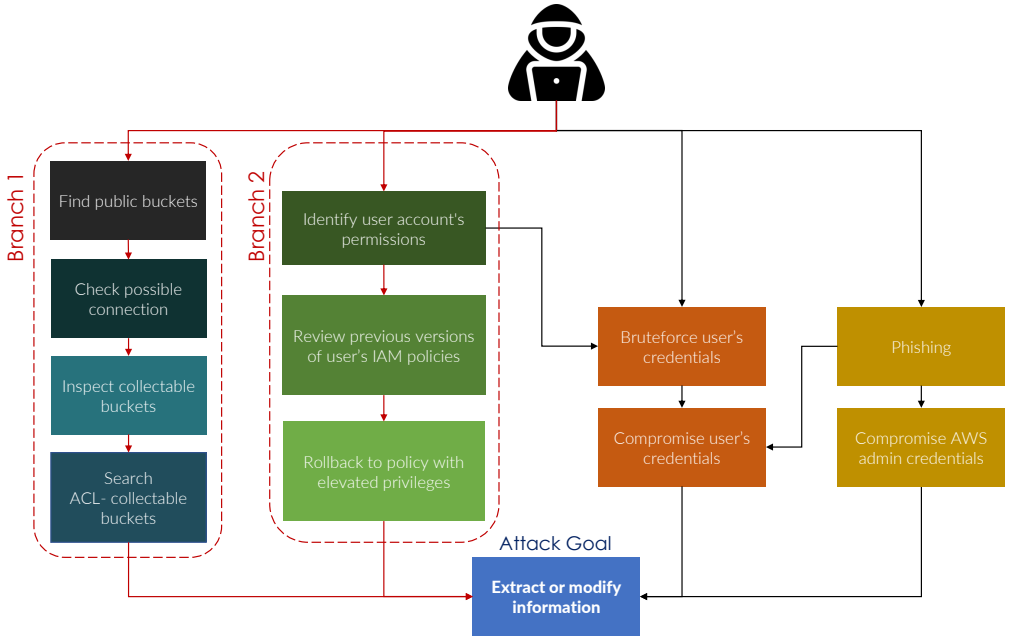

Figure 3. Designed attack tree for the experimental scenario, highlighting the implemented branches.

5.3. Results of ChaosXploit's Execution of Branch 1: Exploitation of Public Buckets

5.3.1. Description

The reason for this experiment is that data can be stored on Amazon S3 and safeguarded from illegal access using encryption techniques and access management software. However, the shared responsibility model of cloud services has caused security configuration errors by the designers of this sort of storage. Exposing the data to the public endangers its availability, confidentiality, and integrity.

Based on the goal of the attack tree (Extract or modify Information), it is possible to define this first experiment following the CE method as follows:

- **Observability**: AWS S3 Buckets that can be found publicly;
- **Steady State**: The buckets to be analyzed suggest having the access controls properly configured;
- **Hypothesis**: If you try to access the objects stored in the buckets, then you will not be able to see their contents or the associated access controls since they are properly configured to prevent information leaks.

Below is a description of how the first branch of the attack tree specified for this scenario was implemented and **executed**. First, by taking regular expressions into account, public buckets were discovered using enumeration approaches. Since Amazon S3 has established some specifications for the bucket names, it is quite simple for an attacker to compile a list of them. Then, boto3, the AWS SDK for Python, was used to carry out the connection check. This stage allowed us to clean up the buckets, removing any that were empty or had incorrect names. Then, ChaosXploit looks at the buckets to see if their objects can be read, and lastly, it checks to see if any buckets provide access to the ACLs.

As shown in Table 3, three monitored variables were considered: (i) **Object-Collectable-Buckets**, which are the buckets that have public files such as pictures, documents, executable files, among others, which may be gathered through the experiment, (ii) **ACL-Collectable-Buckets** which refers to those buckets that have public ACLs, and can be accessed by anyone, and (iii) the **Permissions** obtained from the ACLs.

Table 3. Monitored variables and input parameters considered along the execution of branch 1 by ChaosXploit.

Monitored Variables	
Name	**Description**
Object-Collectable-Buckets	N° of buckets that have public objects and are accessible by anyone
ACL-Collectable-Buckets	N° of buckets that have public ACLs and are accessible by anyone
Permissions	N° of permissions obtained from the ACL.
Input Parameters	
Name	**Description**
Domain (Optional)	Domain name to which you want to identify the buckets
Threads	N° of execution threads
Mode	Object-Collectable-Buckets or ACL-Collectable-Buckets
Output	Output File

Regarding the input values, four were needed to execute the experiment. First, the *domain* is an optional input that should contain the name of the organization to be analyzed. We have considered this option since ChaosXploit can be used as an internal audit tool. Therefore, with this argument, the enumeration of the buckets will be limited to all those that are related to the given domain. In case this input is not provided, ChaosXploit will generate a list of names using brute-force, wordlists, and bucket naming rules defined by AWS. Second, the number of *threads* is considered as an input, so that the process of connecting and reading the buckets' information may be performed in parallel on the different cores, according to the defined thread's value. Third, the *mode* indicates the type of analysis to be performed, whether it aims to find *Object-Collectable-Buckets* or *ACL-Collectable-Buckets*. The last input, *output*, is a file name used to store the results and feed the ChaosXploit continuous validator.

5.3.2. Results Analysis

ChaosXploit's functionality was tested using a list of 3k buckets obtained through a bucket name enumeration process, which can be performed using automated tools.

As seen in the upper left part of Figure 4, all possible actions of the first branch of the attack tree presented in Section 5.2 were executed by ChaosXploit. It is possible to identify that for the second action (Check possible connection), out of the 3k buckets listed, 271 did not allow a connection. This is because the bucket no longer existed or had an invalid name, e.g., it did not follow the common bucket naming characteristics proposed by AWS. This leaves us with 2729 buckets to be tested.

In the case of the third act of the branch (Inspect collectible buckets), 2454 buckets were well configured and passed the steady-state defined in our experiment, since they did not allow reading files or permissions listed in the ACLs. However, 275 did not pass validation.

The lower left part of Figure 4 shows the file extensions that were extracted from the 252 Object-Collectable-Buckets. From each bucket, only the first 50 objects were collected, since some buckets had more than 100,000 files stored, for a total of 7465 collected files. Of all these files it was possible to identify that more than 2000 were images (jpg and png) and approximately 1250 were categorized as others because they could be log files, folders, or had no extension.

To analyze the users and user groups associated with each bucket we first need to know that Amazon S3 has a set of predefined groups:

- **AuthenticatedUsers group** representing all AWS accounts;
- **AllUsers group** allowing anyone in the world to access the resource;
- **LogDelivery group** allowing access logs to be written to the bucket.

Additionally, AWS also defines the following types of permissions:

- **READ** Allows the grantee to list the objects in the bucket;
- **WRITE** Allows the grantee to create new objects in the bucket. For the bucket and object owners of existing objects, it also allows deletions and overwrites of those objects;

- **READ_ACP** Allows the grantee to read the bucket ACL;
- **WRITE_ACP** Allows the grantee to write the ACL for the applicable bucket;
- **FULL_CONTROL** Allows the grantee the READ, WRITE, READ_ACP, and WRITE_ACP permissions on the bucket

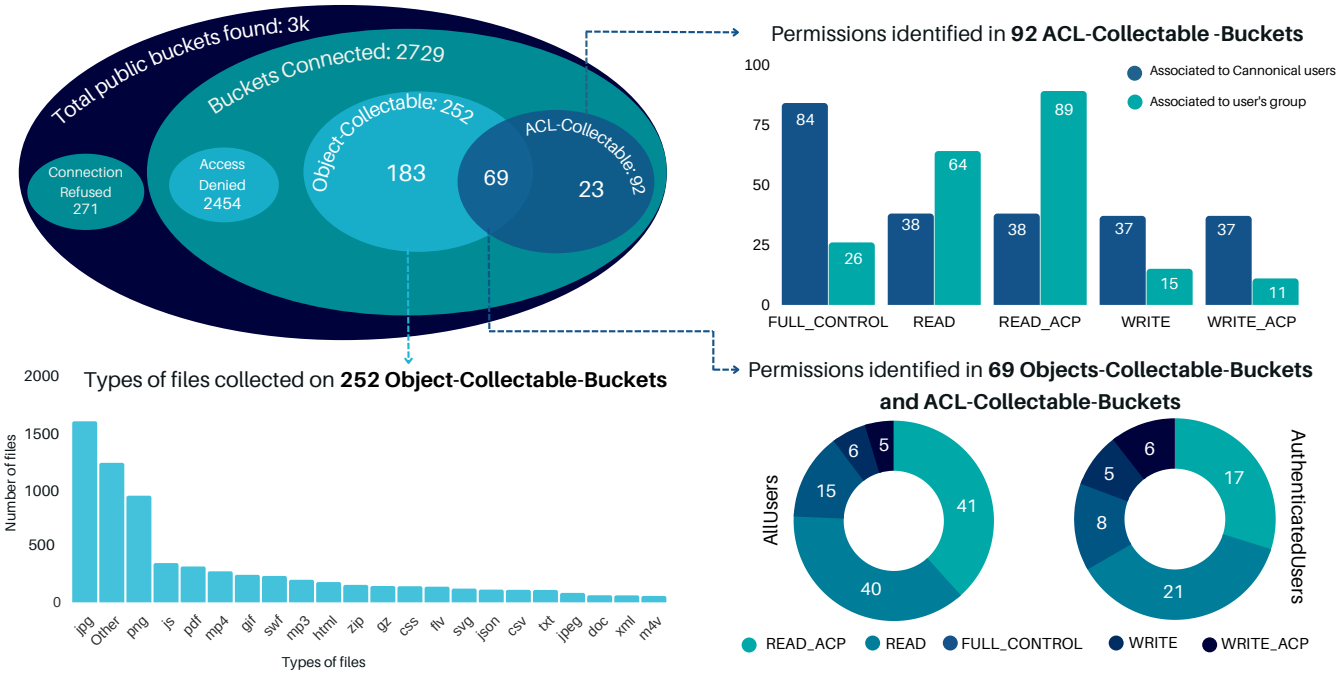

Figure 4. Results of the execution of ChaosXploit to achieve the defined attack goal (extract or modify information) through the branch .

In the upper right part of Figure 4 is possible to identify that 92 of the 257 buckets allowed the extraction of the ACLs. Up to 13 permissions per bucket were identified. Some of them showed information about the user who owned the bucket (known as **CanonicalUser** by AWS); others showed data about the users who belong to one of the predefined groups by AWS and had access to the bucket. Then, it is worth noting that for the information associated with canonical users, the FULL_CONTROL permission was enabled for 84 buckets (91.3%). In the case of the data associated with the users who belong to any of the groups, 64 (69.5%) of them allow the reading of the stored objects (READ permission) and 89 (96.7%) allow the reading of the ACLs (READ_ACP permission).

Finally, we analyze the results of those buckets that allowed the extraction of both objects and ACLs. As seen in the lower right part of Figure 4, 69 buckets (25%) allowed both tasks to be performed. These were filtered by the *AllUsers* and *AuthenticatedUsers* user groups and it was identified that 41 (38.3%) from the *AllUsers* group and 17 (29.8%) from the *AuthenticatedUsers* group were allowed to read the ACLs and the objects. Nevertheless, it was identified that 11 buckets (10.3%) from the *AllUsers* group and 11 buckets (19.3%) from the *AuthenticatedUsers* group allowed the modification of their content (WRITE permission) and the alteration of the ACLs (WRITE_ACP permission), indicating a big flaw that could severely compromise the confidentiality, integrity, and availability of the stored data.

With these results, we have noticed the importance of not only providing a tool for the detection of flaws or vulnerabilities but also seeing it as an aid to infer possible mitigations to prevent the exploitation of such vulnerabilities.

Table 4 shows the summary of the results considering the differences between traditional pentesting and SCE presented in Section 3.3. In this case, it is important to highlight that different tools (s3enum https://github.com/koenrh/s3enum (accessed on 11 October 2022), Sublist3r https://github.com/aboul3la/Sublist3r (accessed on 11 October 2022), bucketkicker https://github.com/craighays/bucketkicker (accessed on 11 October 2022))

may be integrated to ChaosXploit to execute this experiment, which allows us to enumerate the names of the buckets in an optimal way. After the bucket names are identified, ChaosXploit may perform the rest of the actions in a completely automated way. In addition, as we have refuted the hypothesis, ChaosXploit allows us to report a vulnerability related to misconfiguration because the security assumptions on the buckets have not passed the validation of the steady state of the experiment.

Table 4. Results of ChaosXploit's execution of branch 1 in terms of differences between traditional pentesting and SCE.

	SCE
People implementing	Executed by ChaosXploit's team
Methodology behind	Chaos Engineering principles
Security approach	Defensive
Available tools	ChaosXploit
Grade of automatization	All actions to be performed in this branch of the tree have been automated
Expected frequency	By definition, high frequency
Phase of SDLC where applied	Along all the SDLC
Scope of tests	Full test on the buckets list
Kind of vulnerabilities detected	Security assumptions about the configurations of the buckets

5.4. Results of ChaosXploit's Execution of Branch 2: Exploitation of Private Buckets

5.4.1. Description

This second branch refers to scenarios where the AWS policy administration in an organization is not working properly, and a user account maintains unnecessary policies, e.g., when a user changes role or area in a company. This scenario, caused by a misconfiguration in the IAM module, may be more critical when such a policy enables the user account to restore policies. Thus, the user may cause an elevation of privileges that allow him/her access to services and data in an unauthorized way. As part of the security inspection that a cybersecurity team could execute over a business infrastructure, one may assume that an internal attacker, e.g., an employee or contractor, could be interested in validating if his/her account allows the execution of policies additional to the required ones for the role. In addition, in the case of an external attacker, he/she could be interested in validating if some previously compromised AWS account, which contains limited permissions, can be elevated.

Considering the previous scenario, the following SCE definitions aligned to the scientific method are posed:

- **Observability**: List and status of the policies assigned to an AWS user account under analysis;
- **Steady State**: The AWS user account under analysis has policies assigned to him/her according to minimum privilege and need-to-know policies specific to his/her role in the organization;
- **Hypothesis**: Policies assigned to an AWS user account should not be modified in an unauthorized way.

For the **execution** of the second branch of the tree, ChaosXploit checks the policies assigned to the user account's profile defined for the experiment setup. If it identifies that the user account has the permission to restore previous versions of its policies, then it lists all the policy versions and searches for the one with elevated permissions to gain access to a privileged service, i.e., the AWS managed storage service (S3). This will achieve the goal of the attack tree: to extract or modify information. If the user does not have such a permission, ChaosXploit will start the execution of the third branch of the presented attack tree.

The upper part of Table 5 shows the two main variables that were monitored through the experiments of branch 2, i.e., *Attached-User-Policies* and *Current-Policy*. First, *Attached-User-Policies* is used at two moments of the branch execution: (i) at the beginning of branch 2 to identify all policies associated with a user account, and (ii) at the middle of branch 2 to

identify permission associated with the user account that allows for the restoration of the previous version of policies and a previous policy that may be a suitable candidate to be restored, e.g., a policy that allows for the extraction and modification of information in the AWS S3 service. Second, *Current-Policy* represents the current version of the user's policy set, so this variable verifies whether the previous policy's restoration was successful.

Table 5. Monitored variables and input parameters considered along the execution of branch 2 by ChaosXploit.

Monitored Variables	
Name	**Description**
Attached-User-Policies	Listing of policies assigned to a user
Current-Policy	Policy currently assigned to the user
Input Parameters	
Name	**Description**
User-Account	User account from which the actions will be performed
Output	Output file

On the other hand, the lower part of Table 5 shows the input elements that ChaosXploit receives for the execution of this branch. In this case, ChaosXploit uses the name of the user account (user account) for whom the security inspection must be performed. In addition, ChaosXploit takes as a parameter the name of the output file (*output*) for where to store the results.

5.4.2. Results Analysis

Figure 5 shows the execution of ChaosXploit for branch 2, which includes (i) the setup of ChaosXploit (lines 1–5), (ii) the steady state validation which assumes a correct configuration of the policies assigned to the user account under analysis (lines 6–10), (iii) execution of the actions that allow validating the hypothesis through an attempt to restore a previous policy (lines 11–20). This last set of lines includes listing the user policies (line 13–14), validating the current version (line 15), identifying the version that allows the privilege escalation (line 16), restoring the desired policy (line 17–18) and validation of the restore (line 20).

```
1  [2022-08-11 18:55:02 INFO] Validating the experiment's syntax
2  [2022-08-11 18:55:02 INFO] Experiment looks valid
3  [2022-08-11 18:55:02 INFO] Running experiment: Policy Rollback
4  [2022-08-11 18:55:02 INFO] Steady-state strategy: default
5  [2022-08-11 18:55:02 INFO] Rollbacks strategy: default
6  [2022-08-11 18:55:02 INFO] Steady state hypothesis: User's policy should be well configured
7  [2022-08-11 18:55:02 INFO] Probe: Checking configurations
8  Is Steady State validated?: True
9  Steady State validated
10 [2022-08-11 18:55:02 INFO] Steady state hypothesis is met!
11 [2022-08-11 18:55:02 INFO] Playing your experiment's method now...
12 [2022-08-11 18:55:02 INFO] Action: 1. Performing Rollback
13 Listing Attached User Policies
14 Getting all versions
15 v1 is current version
16 Version v2 has full access
17 Trying policy rollback
18 !! Rollback successfull !!
19
20 v2  is current version
```

Figure 5. Validation of the steady state and elevation of privileges achieved by ChaosXploit through branch 2.

Table 6 shows the details of each of the policy versions found by ChaosXploit for the user account under analysis. This table lists the policy versions, the effects on the actions (either allow or deny access), the actions that indicate what the user can or cannot do, the resources on which the action may be applied, and additional conditions under which the policy has an effect. The current policy version (1) has limited actions related to the IAM service, but it still allows to change the policy through the action *SetDefaultPolicyVersion*.

It is also possible to identify the policy version 5, which includes some actions to manage the AWS S3 service. However, such actions would not allow reaching the attack goal because they do not allow modifying information. Finally, the version chosen by ChaosXploit (2) to be restored was the one that allows any action on any resource without any condition.

Table 6. Policy versions found by ChaosXploit through branch 2.

Version	Effect	Action(s)	Resource(s)	Condition
1 (Current)	Allow	"iam:Get*", "iam:List*", "iam:SetDefaultPolicyVersion"	*	None
2	Allow	*	*	None
3	Deny	*	*	IP Condition
4	Allow	"iam:Get*"	*	Time Condition
5	Allow	"s3:ListBucket", "s3:GetObject", "s3:ListAllMyBuckets"	*	None

Once the previous policy is restored, as shown in Figure 5, ChaosXploit initiates the actions shown in Figure 6. Between the first actions, ChaosXploit establishes the connection to the target and defines the *collect* mode to inspect the files in the bucket and the *write* mode to write a new file (lines 1–4). Additionally, ChaosXploit creates new files in the S3 bucket, as this experiment was being executed in its own controlled environment (lines 5–6). The validation of the steady state at lines 8–10 failed in this case as the policy settings can be manipulated and used to alter the information.

```
1  [2022-08-11 18:55:05 INFO] Action: 2. Inspecting Buckets
2  All tests will be executed in anonymous mode
3
4  Starting modes: ['collect', 'write']
5  Checking bucket chaosxploit-bucket
6  Success: bucket 'chaosxploit-bucket' allows for uploading arbitrary files!!!
7  Bucket 'chaosxploit-bucket' collectable: http://s3.us-east-1.amazonaws.com/chaosxploit-bucket/file.txt !!!
8  [2022-08-11 18:55:08 INFO] Steady state hypothesis: User's policy should be well configured
9  [2022-08-11 18:55:08 INFO] Probe: Checking configurations
10 Is Steady State validated?: False
11 Failed validation
12 [2022-08-11 18:55:08 CRITICAL] Steady state probe 'Checking configurations' is not in the given tolerance so failing this experiment
13 [2022-08-11 18:55:08 INFO] Experiment ended with status: deviated
14 [2022-08-11 18:55:08 INFO] The steady-state has deviated, a weakness may have been discovered
```

Figure 6. Attack goal (extract or modify information) achieved by ChaosXploit through branch 2.

In experiments executed along branch 1 (Section 5.3) and branch 2 (Section 5.4), the attack goal was achieved so the experiments ended in a **critical** state similar to the one seen in line 11 at Figure 6. Table 7 shows the summary of the results for this second experiment, considering the differences between traditional pentesting and SCE presented in Section 3.3. In this case, we highlight the ChaosXploit capabilities to develop this kind of experiment that exploits the AWS authorization module. Additionally, we define the scope of the experiment only to users belonging to the same IAM account. Finally, as the experiment ended in a critical state, we report a vulnerability associated with privilege escalation, which allows a user to pass from few to many permissions, putting the confidentiality and integrity of the information available in the different AWS services at risk.

Table 7. Results of ChaosXploit's execution of branch 2 in terms of the differences between traditional pentesting and SCE.

	SCE
People implementing	Executed by ChaosXploit's team
Methodology behind	Chaos Engineering principles
Security approach	Defensive
Available tools	ChaosXploit
Expected frequency	By definition, high frequency
Phase of SDLC where applied	Along all the SDLC
Scope of tests	Users belonging to the IAM account
Kind of vulnerabilities detected	Privilege escalation considering the policy versions assigned to users

6. Toward an Adoption of SCE in Industry

With the growing adoption of CE, many companies have included it as a discipline for improving reliability. According to InfoQ [41], the appropriation of CE practices to inject failures and generate resilience has evolved to the "Early Majority stage", which means that its adoption is about one-third of the overall population. Gremlin, Litmus, and Steadybit are some key CE initiatives that have contributed to this achievement.

The stories of the adoption of CE reported by companies such as Capital One, Linkedin, Google, and Microsoft [34] are examples of its wide acceptance. The appropriation of CE as a common discipline to inject failures and generate resilience provides arguments to justify the success of this discipline between industry and academia.

Not only have the failures of the infrastructure attracted the attention of practitioners, but data breaches and security incidents have risen in recent years [42]. Failure to implement basic configurations and appropriate security controls have led to causes that contribute to the security incidents. Undoubtedly organizations are being asked to produce with extremely high throughput and with very little resources to maintain the security status quo. All the while, there is a divergent gap in how we design and build distributed systems and approach security engineering.

In this sense, SCE serves as a foundation for developing a learning culture around how organizations build, operate, instrument, and secure their systems. The goal of these experiments is to move security in practice from subjective assessment into objective measurements. As they do in CE, Security Chaos experiments allow security teams to reduce the "unknown unknowns" and replace "known unknowns" with information that can drive improvements to security posture. The promise in terms of adoption and sophistication is immense.

Even though introducing false positives into production networks and other infrastructures under the context of CE is a common practice nowadays, SCE is still seen as more of an academic research topic than industry practice. Nevertheless, in recent years, SCE is starting to become known in the industry. One example is the Thoughtworks report [43], which documented an evolution around this technique migrating SCE from a phase of "Assess" to "Trial", which means that SCE could be eventually used in a controlled way and validated that the security policies in place are robust enough to handle common security failure modes.

Another remarkable example of the application of SCE in the industry was documented by Jamie Dicken [44]. She wrote about her SCE journey at Cardinal Health, a global Fortune 20 healthcare manufacturer and distributor of medical and laboratory products and a provider of performance and data solutions for healthcare facilities. Cardinal Health needed an applied security model to protect critical infrastructure and data as it was moving to the cloud, and SCE became the most appropriate answer. Cardinal Health created a process named Continuous Verification and Validation (CVV) that, by using SCE, allowed them to continuously verify that security controls were working correctly and as expected.

Adopting SCE first requires a solid understanding of the principles of chaos. For example, insufficient observability of the chaotic experiments would impede drawing reliable statements about a hypothesis. After understanding the fundamentals, the next step should start by developing competency and confidence in the methods and tools needed to perform the SCE experiments. For this, a new SCE practitioner may decide to start designing small and manual experiments. In case the hypothesis is not disproved, we can automate the experiment. Here, ChaosXploit may play a key role as one of the few SCE platforms existing nowadays that may enable the industry to design and execute experiments aimed at the automatic and controlled exploitation of vulnerabilities and validation of systems security. Security validations can also be achieved progressively through security chaos game days that allow players to advance in this path without causing a security incident on production.

On the side, diverse teams should know and try SCE since it is no longer a limited concept for Security Engineers or security teams. We believe that if SCE begins as an

engineering practice, it could be quickly adopted by other roles (Cloud Engineers, Software Engineers, Site Reliability Engineers) and teams (platform, infrastructure, operations, and application development) as it would allow them to improve the reliability of their applications through proactive testing of their own security.

7. Conclusions and Future Work

The digital revolution, or digital transformation, as it has been called in recent years, has proven to be an incredible driving factor in our society. Thanks to this revolution, our society was able to handle some of the most serious restrictions that the recent pandemic put on different essential services, e.g., the use of highly interactive e-health services in response to the restrictions regarding in-person medical consultations, exploitation of e-learning platforms to face the limitations in the physical access to formal educational services, enabling e-payments as an alternative to the use of traditional financial services, among many others.

On the downside, such a change also implies the existence of ill-motivated entities that constantly try to attack connected systems to damage the confidentiality, integrity, or availability of the provided online services. Such threat entities use increasingly advanced techniques, for example, based on malware campaigns [45] or threats addressed to a specific technology [46].

Over the last years, a novel paradigm has emerged, the so-called Chaos Engineering (CE), whose main objective consists of testing the resiliency of distributed and complex systems through continuous observation and experimentation. More recently, the paradigm has evolved to embrace the entire cybersecurity ecosystem, i.e., Security Chaos Engineering (SCE) comes into play to defend the system assets against cyberattacks through continuous and rigorous experimentations on possible security holes and consequent mitigations.

In this paper, we proposed ChaosXploit, an SCE-powered framework that can conduct SCE experiments on different target architectures. Based on the hypothesis generated by the knowledge database and the attack representations, ChaosXploit executes SCE experiments over a target to find a potential security problem as an ultimate goal. In addition, ChaosXploit features an observer that is in charge of verifying the change between the steady state of a certain hypothesis and the current state of the system. To prove the capabilities of ChaosXploit, a set of experiments was conducted on several AWS S3 buckets, evaluating their security characteristics with SCE. The results demonstrated that our approach could be successful, highlighting several unprotected buckets for a specific attack path. To foster its adoption, ChaosXploit was made publicly available for the cybersecurity community through the repository of the project [39].

Future work will explore the possibility of widening the ChaosXploit framework target architectures to include other use cases, systems, or providers. That is, the extension of the Attack Trees knowledge base is considered mandatory to include a number of different application scenarios, which can lead to the potential improvements of ChaosXploit, too. Particularly, one could easily argue that using a standardized attack modeling methodology (e.g., MITRE ATTC&K [47]) would be beneficial for the proposed SCE framework, even if some adjustments are needed to achieve full compliance. Besides, integrating a recommendation module to suggest countermeasures once a security flaw is discovered is worth investigating. In this sense, several attack models have been proposed in the literature so far, and some of them already integrate the Attack Trees representation adding countermeasures (e.g., Attack Countermeasures Trees [48], Attack Response Trees [49], etc.). Thus, ChaosXploit may incorporate those representations in the Knowledge base and select the optimal reaction to fire against the threat based on specific criteria [50]. Moreover, the performance of ChaosXploit should be further evaluated to prove its usefulness in performance-demanding or critical scenarios. Expressly, the assessment of the response time and resource consumption is essential to argue the applicability of the presented framework in scenarios where the threat discovery procedure must be executed in real-time or with limited computation capabilities.

Author Contributions: Conceptualization, S.P.C., P.N. and D.D.-L.; methodology, S.P.C., P.N. and D.D.-L.; software, S.P.C.; validation, P.N. and D.D.-L.; formal analysis, S.P.C., P.N. and D.D.-L.; investigation, S.P.C. and P.N.; resources, D.D.-L.; data curation, S.P.C.; writing—original draft preparation, S.P.C., P.N., D.D.-L. and Y.N.R.; writing—review and editing, P.N. and D.D.-L.; visualization, S.P.C.; supervision, P.N. and D.D.-L.; project administration, D.D.-L.; funding acquisition, D.D.-L. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been supported by Universidad del Rosario (Bogotá) through the project "IV-TFA043—Developing Cyber Intelligence Capacities for the Prevention of Crime" and through "Becas para Estancias de Docencia e Investigación. Universidad del Rosario".

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

SCE	Security Chaos Engineering
ICT	Information and Communication Technology
LEA	Law Enforcement Agencies
SRE	Site Reliability Engineering
SLI	Service Level Indicator
SLO	Service Level Objective
CE	Chaos Engineering
SPL	Software Product Line
API	Application Programming Interface
AWS	Amazon Web Services
GCP	Google Cloud Platform
CVSS	Common Vulnerability Scoring System
OWASP	Open Web Application Project
SoS	System of Systems
VUAV	Virtual Unmanned Aerial Vehicle
CRD	Custom Resource Definition
CTK	ChaosToolKit
ACL	Access Control List
SDK	Software Development Kit
CVV	Continuous Verification and Validation

References

1. Rodríguez, J.I.; Durán, S.R.; Díaz-López, D.; Pastor-Galindo, J.; Mármol, F.G. C3-Sex: A Conversational Agent to Detect Online Sex Offenders. *Electronics* **2020**, *9*, 1779. [CrossRef]
2. Sánchez, P.; Huertas, A.; Bovet, G.; Martínez, G.; Stille, B. An ML and Behavior Fingerprinting-based Framework for Cyberattack Detection in IoT Crowdsensing Platforms. In Proceedings of the VII Jornadas Nacionales de Investigación en Ciberseguridad (JNIC), Bilbao, Spain, 27–29 June 2022; Volume 1, pp. 188–191.
3. Botello, J.V.; Mesa, A.P.; Rodríguez, F.A.; Díaz-López, D.; Nespoli, P.; Mármol, F.G. BlockSIEM: Protecting Smart City Services through a Blockchain-based and Distributed SIEM. *Sensors* **2020**, *20*, 4636. [CrossRef] [PubMed]
4. Díaz-López, D.; Dólera-Tormo, G.; Gómez-Mármol, F.; Martínez-Pérez, G. Managing XACML systems in distributed environments through Meta-Policies. *Comput. Secur.* **2015**, *48*, 92–115. [CrossRef]
5. Useche-Peláez, D.E.; Sepúlveda-Alzate, D.; Díaz-López, D.O.; Cabuya-Padilla, D.E. Building malware classificators usable by State security agencies. *Iteckne* **2018**, *15*, 107–121. [CrossRef]
6. Pastor-Galindo, J.; Sáez, R.; Maestre, J.; Sotelo, M.; Gómez, F.; Martínez, G. Designing a platform for discovering TOR onion services. In Proceedings of the VII Jornadas Nacionales de Investigación en Ciberseguridad (JNIC), Bilbao, Spain, 27–29 June 2022; Volume 1, pp. 30–33.

7. Beyer, B.; Jones, C.; Petoff, J.; Murphy, N.R. *Site Reliability Engineering: How Google Runs Production Systems*, 1st ed.; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2016.
8. Beyer, B.; Murphy, N.; Rensin, D.; Kawahara, K.; Thorne, S. *The Site Reliability Workbook: Practical Ways to Implement SRE*; O'Reilly Media: Sebastopol, CA, USA, 2018.
9. Principles of Chaos Engineering. Available online: https://principlesofchaos.org/ (accessed on 9 November 2022).
10. Pawlikowski, M. *Chaos Engineering: Site Reliability through Controlled Disruption*; Manning: Shelter Island, NY, USA, 2021.
11. Díaz-López, D.; Blanco Uribe, M.; Santiago Cely, C.; Tarquino Murgueitio, D.; Garcia Garcia, E.; Nespoli, P.; Gómez Mármol, F. Developing Secure IoT Services: A Security-Oriented Review of IoT Platforms. *Symmetry* **2018**, *10*, 669. [CrossRef]
12. Díaz-López, D.; Dólera Tormo, G.; Gómez Mármol, F.; Alcaraz Calero, J.M.; Martínez Pérez, G. Live digital, remember digital: State of the art and research challenges. *Comput. Electr. Eng.* **2014**, *40*, 109–120. [CrossRef]
13. Torkura, K.A.; Sukmana, M.I.; Cheng, F.; Meinel, C. CloudStrike: Chaos Engineering for Security and Resiliency in Cloud Infrastructure. *IEEE Access* **2020**, *8*, 123044–123060. [CrossRef]
14. Palacios, S.; Díaz-López, D.; Nespoli, P. ChaosXploit: A Security Chaos Engineering framework based on Attack Trees. In Proceedings of the VII Jornadas Nacionales de Investigación en Ciberseguridad (JNIC), Bilbao, Spain, 27–29 June 2022; Volume 1, pp. 130–137.
15. Basiri, A.; Behnam, N.; de Rooij, R.; Hochstein, L.; Kosewski, L.; Reynolds, J.; Rosenthal, C. Chaos Engineering. *IEEE Softw.* **2016**, *33*, 35–41. [CrossRef]
16. Camacho, C.; Cañizares, P.C.; Llana, L.; Núñez, A. Chaos as a Software Product Line—A platform for improving open hybrid-cloud systems resiliency. In *Software—Practice and Experience*; Wiley: Hoboken, NJ, USA, 2022; pp. 1–34. [CrossRef]
17. Simonsson, J.; Zhang, L.; Morin, B.; Baudry, B.; Monperrus, M. Observability and chaos engineering on system calls for containerized applications in Docker. *Future Gener. Comput. Syst.* **2021**, *122*, 117–129.
18. Jernberg, H.; Runeson, P.; Engström, E. Getting started with chaos engineering—Design of an implementation framework in practice. In Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'20), Bari, Italy, 5–9 October 2020; Volume 1211704, p. 10. [CrossRef]
19. Zhang, L.; Morin, B.; Haller, P.; Baudry, B.; Monperrus, M. A Chaos Engineering System for Live Analysis and Falsification of Exception-Handling in the JVM. *IEEE Trans. Softw. Eng.* **2021**, *47*, 2534–2548.
20. ChaoSlingr: Introducing Security into Chaos Testing. Available online: https://github.com/Optum/ChaoSlingr (accessed on 9 November 2022).
21. Rinehart, A.; Shortridge, K. *Security Chaos Engineering Gaining Confidence in Resilience and Safety at Speed and Scale*; Technical Report; O'Reilly Media: Sebastopol, CA, USA, 2021.
22. Torkura, K.A.; Sukmana, M.I.; Cheng, F.; Meinel, C. Security Chaos Engineering for Cloud Services: Work in Progress. In Proceedings of the 2019 IEEE 18th International Symposium on Network Computing and Applications, NCA 2019, Cambridge, MA, USA, 26–28 September 2019. [CrossRef]
23. Torkura, K.A.; Sukmana, M.; Cheng, F.; Meinel, C. Continuous auditing and threat detection in multi-cloud infrastructure. *Comput. Secur.* **2021**, *102*, 102124. [CrossRef]
24. Sharieh, S.; Ferworn, A. Securing APIs and Chaos Engineering. In Proceedings of the 2021 IEEE Conference on Communications and Network Security (CNS), Tempe, AZ, USA, 4–6 October 2021; pp. 290–294. [CrossRef]
25. Bailey, T.; Marchione, P.; Swartz, P.; Salih, R.; Clark, M.; Denz, R. Measuring resiliency of system of systems using chaos engineering experiments. In Proceedings of the 2022 SPIE 12117, Disruptive Technologies in Information Sciences VI, Orlando, FL, USA, 3–7 April 2022; Volume 1211704, p. 26. [CrossRef]
26. Pierce, T.; Schanck, J.; Groeger, A.; Salih, R.; Clark, M.R. Chaos engineering experiments in middleware systems using targeted network degradation and automatic fault injection. In Proceedings of the Open Architecture/Open Business Model Net-Centric Systems and Defense Transformation 2021, Online, 12–17 April 2021; Suresh, R., Ed.; International Society for Optics and Photonics, SPIE: Bellingham, WA, USA, 2021; Volume 11753, p. 117530A. [CrossRef]
27. The Netflix Simian Army. Available online: https://netflixtechblog.com/the-netflix-simian-army-16e57fbab116 (accessed on 14 March 2022).
28. Gremlin. Available online: https://www.gremlin.com/ (accessed on 10 November 2022).
29. Chaos Mesh. Available online: https://chaos-mesh.org/ (accessed on 10 November 2022).
30. Litmus. Available online: https://litmuschaos.io/ (accessed on 10 November 2022).
31. ChaosToolkit. Available online: https://chaostoolkit.org/ (accessed on 10 November 2022).
32. Chaos Engineering: The History, Principles, and Practice. Available online: https://www.gremlin.com/community/tutorials/chaos-engineering-the-history-principles-and-practice/ (accessed on 21 March 2022).
33. UnitedHealthGroup. Available online: https://www.unitedhealthgroup.com/ (accessed on 14 March 2022).
34. Rosenthal, C.; Jones, N. *Chaos Engineering: System Resiliency in Practice* ; O'Reilly Media: Sebastopol, CA, USA, 2020.
35. Verica. Available online: https://www.verica.io/ (accessed on 14 March 2022).
36. Nespoli, P.; Papamartzivanos, D.; Mármol, F.G.; Kambourakis, G. Optimal Countermeasures Selection Against Cyber Attacks: A Comprehensive Survey on Reaction Frameworks. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 1361–1396. [CrossRef]
37. Raj, S.; Walia, N.K. A Study on Metasploit Framework: A Pen-Testing Tool. In Proceedings of the 2020 International Conference on Computational Performance Evaluation (ComPE), Shillong, India, 2–4 July 2020; pp. 296–302. [CrossRef]

38. FOCA (Fingerprinting Organizations with Collected Archives). Available online: https://github.com/ElevenPaths/FOCA (accessed on 14 March 2022).
39. ChaosXploit. Available online: https://github.com/SaraPalaciosCh/ChaosXploit (accessed on 10 November 2022).
40. Rapid7. *2021 Cloud Misconfiguration Report*; Rapid7: Boston, MA, USA, 2021.
41. Wiggers, S.J. DevOps and Cloud InfoQ Trends Report. Available online: https://www.infoq.com/articles/devops-and-cloud-trends-2022/ (accessed on 10 November 2022).
42. *2018 Cost of Data Breach Study: Impact of Business Continuity Management*; Technical Report; Benchmark research sponsored by IBM; Ponemon Institute LLC: Traverse City, MI, USA, 2018.
43. ThougthWorks. Security Chaos Engineering. Available online: https://www.thoughtworks.com/radar/techniques/security-chaos-engineering (accessed on 10 November 2022).
44. Rinehart, A.; Shortridge, K.; Safari, a.O.M.C. *Security Chaos Engineering*; O'Reilly Media, Incorporated: Sebastopol, CA, USA, 2020.
45. Martínez Martínez, I.; Florián Quitián, A.; Díaz-López, D.; Nespoli, P.; Gómez Mármol, F. MalSEIRS: Forecasting Malware Spread Based on Compartmental Models in Epidemiology. *Complexity* **2021**, *2021*. [CrossRef]
46. Nespoli, P.; Díaz-López, D.; Gómez Mármol, F. Cyberprotection in IoT environments: A dynamic rule-based solution to defend smart devices. *J. Inf. Secur. Appl.* **2021**, *60*, 102878. [CrossRef]
47. Ahmed, M.; Panda, S.; Xenakis, C.; Panaousis, E. MITRE ATT&CK-Driven Cyber Risk Assessment. In Proceedings of the 17th International Conference on Availability, Reliability and Security, Vienna, Austria, 23–26 August 2022. [CrossRef]
48. Roy, A.; Kim, D.S.; Trivedi, K.S. Attack countermeasure trees (ACT): Towards unifying the constructs of attack and defense trees. *Secur. Commun. Netw.* **2012**, *5*, 929–943.
49. Zonouz, S.A.; Khurana, H.; Sanders, W.H.; Yardley, T.M. RRE: A Game-Theoretic Intrusion Response and Recovery Engine. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 395–406. [CrossRef]
50. Nespoli, P.; Mármol, F.G.; Vidal, J.M. A Bio-Inspired Reaction Against Cyberattacks: AIS-Powered Optimal Countermeasures Selection. *IEEE Access* **2021**, *9*, 60971–60996. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.