ChaosXploit: A Security Chaos Engineering framework based on Attack Trees

Author
Sara Palacios Chavarro

Work presented as a requirement to qualify for the
degree of Professional in Applied Mathematics and Computer Science

Director
Daniel Orlando Díaz López

School of Engineering, Science and Technology
Applied Mathematics and Computer Science
Universidad del Rosario

Bogotá - Colombia
2022

Contents

1 Abstract

Security incidents may have several origins. However, many times they are caused due to components that are supposed to be correctly configured or deployed. That is, traditional methods may not detect those security assumptions, and new alternatives need to be tried. Security Chaos Engineering (SCE) represents a new way to detect such failing components in order to protect assets under cyber risk scenarios.

To demonstrate the application of SCE in security, this degree project presents, in the first place, an introduction to the fundamentals of Chaos Engineering (CE) as SCE inherits its principles and methodology. This is done to understand its application in engineering, a series of analyses of the proposed frameworks and tools for the implementation of CE is provided, and its functionality is tested with four experiments.

In the second place, this degree project proposes ChaosXploit, a security chaos engineering framework based on attack trees, which leverages the CE methodology along with a knowledge database composed of attack trees to detect and exploit vulnerabilities in different targets as part of an offensive security exercise. Once the theoretical and conceptual components of SCE are detailed and the proposal for ChaosXploit is explained, a set of experiments are conducted to validate the feasibility of ChaosXploit to validate the security of cloud managed services, i.e. Amazon buckets, which may be prone to misconfigurations.

2 Introduction

Site Reliability Engineering (SRE) is defined as a discipline focused on improving systems' design and operation to make them more scalable, reliable, and efficient [3].With this in mind, by 2006 Google founded the Disaster Recovery Testing (DiRT) program to test the resilience of systems by regularly simulating various internal system failures[1]. These activities led to the creation of a new approach for testing the resiliency of distributed systems [2], which is known as **Chaos Engineering** (CE). CE is used to identify the system's immunities when damage is injected, so vulnerabilities can be found and lately mitigated. CE tests are designed to "build confidence in the system's capability to withstand turbulent conditions in production" [13].

Designing CE experiments implies defining a prepared and controlled environment to analyze a target system [20] and applying a scientific method that allows to observe the environment, define a set of hypotheses, and validate them. CE has proven to be extremely useful in validating attributes of reliability and availability in a production environment. Nevertheless, it may not be enough if the final objective is an holistic validation of the security level of the system as is required for different distributed systems, e.g. secure IoT services [10] or personal data managers with high security requirements [11].

Considering what was previously said, some efforts have come up towards applying CE to cybersecurity in the last five years, which is known as **Security Chaos Engineering** (SCE). In particular, SCE aims to use the CE principles to evaluate the three most important attributes of a system from an holistic cybersecurity perspective, i.e., confidentiality, integrity, and availability [28].

Noting that this new methodology can have a great impact on new developments by reducing vulnerabilities through experimentation, it has been decided for this degree project to follow this innovative line to provide a new CE security framework based on attack trees, known as ChaosXploit.

This document is structured as follows: First, Section 3 presents the context and the main problem addressed in this work. Then, Section 4 presents the main objective

of the work along with the specific ones. Next, Section 5 presents the methodology that was followed in this thesis and the inner activities developed in the two phases of this project. Section 6 collects the most recent works on CE and SCE exploring its advantages and disadvantages. Further, Sections 7 and 8 narrate the theoretical fundamentals of CE, as well as the detailed execution of four CE experiments that allowed the appropriation of the concepts of this subject. Later, Section 9 shows the theoretical component on SCE and the possible contribution opportunities. In Sections 10 and 11 the proposal of this degree work that is ChaosXploit is presented and validated. Finally, Section 13 concludes the work, showing some future works that can improve the proposal.

3 Problem Description

As information technologies and the fields of study related grow, so do the techniques used by adversaries to search for vulnerabilities in applications and exploit them [12]. To avoid these vulnerabilities being exposed when the applications are already deployed, it is necessary to think about security from the first phases of the software development cycle.

However, there are organizations that claim that their developers do not have the necessary knowledge about security or the necessary tools to implement it [15]. In addition, an insecure application could jeopardize the three fundamental pillars of cybersecurity: confidentiality, integrity, and availability.

To prevent the fundamental pillars of cybersecurity from being breached, OWASP [18], a foundation working to improve software security, lists the 10 most common computer security vulnerabilities [19]. This set of vulnerabilities aims to provide organizations with the techniques most commonly used by modern attackers and the appropriate mitigation processes so that they can use them to minimize the risks to their systems.

These techniques provided by OWASP are commonly used in Static (SAST) or Dynamic (DAST) Analysis Security Testing[15]. However, performing this type of

testing can be a very exhaustive process and it is usually necessary to hire an external party to implement them.

In response to this problem, there is an emerging field of study that seeks to identify flaws in security controls to defend against malicious conditions [22], known as SCE. This field proposes a methodology that complements the incorporation of safety within the software development cycle. However, it is a concept that has been developing over the last five years, and the few tools that exist for its implementation have been archived or are not open source.

Therefore, this area is considered worthy of study to propose a new framework to support the process of vulnerability detection and mitigation during the testing, deployment and maintenance phases of the software development cycle and thus reinforce traditional methodologies.

4 Objectives

The main objective of this degree work is to design and implement a SCE framework based on attack trees to support the process of identifying vulnerabilities in a system in the testing, deployment, and maintenance phases. Developing this objective will allow not only to understand the theoretical basis of CE but also to apply its methodology and ideology to traditional methodologies specifically in pentesting exercises for those phases of the software development life cycle where security controls must be involved. To achieve this, the following specific objectives were set:

- Review the theoretical and conceptual approaches to CE and SCE.
- Comparatively analyze tools that have applied chaos engineering in real contexts and identify opportunities for improvement in their implementation.
- Propose an SCE framework that uses attack trees as the main flowchart for attack execution.

- Conduct experiments to test the effectiveness of the framework on different targets.

5 Methodology

The development of this degree work was divided into an exploratory and an experimental phase. During the first phase, carried out during the second semester of 2021, the first two specific objectives previously proposed were completed. The first one aimed to review the theory and concepts surrounding CE, its components, and limitations. This was done through a literature review in specialized academic and scientific sources published in recent years, the results are outlined in Sections 6 and 7. In addition, several experiments described in Section 8 were replicated to understand the methodology and explore new ways of implementation.

The second objective focused on the analysis of CE tools and the identification of opportunities for improvement in their implementation. This analysis took the documentation of tools such as *gremlin*[1], *ChaosToolkit*[2], among others to know the capabilities of each one of them. Additionally, the application of security to CE, the motivation to implement it and the possible opportunities to work and contribute were studied and analyzed in Section 9.

The experimental phase was carried out in the first half of 2022. During this phase, the development of ChaosXploit, the framework proposed for this degree project, was carried out. With the analysis of tools, the main characteristics of each one of them were detected and the architecture defined in Section 10 was built. Once the architecture was understood, its functionality was tested with the experiment described in Section 11.

[1] https://www.gremlin.com/
[2] https://chaostoolkit.org/

6 State of the art

As far as CE research is concerned, there are various publications that highlight its capabilities, methodology and definitions. This whole world starts with *Chaos Monkey*, a tool created by Netflix [24] to test the reliability of cloud infrastructure, specifically for Amazon Web Services (AWS).

Following the cloud-native software line, Camacho *et al.* [9] presents *Pystol*, a novel reference architecture for executing fault injection actions focused on cloud-native environments. This architecture is based on the software product line paradigm, which is a reference architecture for building different products that share common software artifacts in a prescribed manner. An empirical study is presented which seeks to analyze how the execution of fault injection actions impacts system behavior when applications and user services are serving requests to test the architecture. This is mounted on a Kubernetes cluster mounted on AWS.

Moreover, Zhang *et al.* [30] proposed *ChaosMachine*, a CE system to analyze the error handling response of a java application put into production. This system is able to reveal the resilience of strengths and weaknesses of each try-catch block with two types of experiments: *falsification experiments* that validate or refute the hypothesis about the behavior of a try-catch block and *exploration experiments* that seek to monitor the behavior of try-catch blocks under perturbation. To test its functionality, 3 real-life projects developed in java were used: TTorrent, BroadLeaf, and X-Wiki. ChaosMachine was able to produce actionable reports for developers to gain more confidence about the resilience of their systems.

In 2021, after developing ChaosMachine, Zhang *et al.* [29] proposed a new framework that performs error injection at the system call level. This framework is known as *PHOEBE* and allows developers to have full observability of system call invocation. *PHOEBE* synthesizes a series of error injection models that systematically amplify natural errors that occur in a production system. Their functionality is tested on two real-life applications: Hedwig and TTorrent. They conclude that it is possible to detect reliability weaknesses with low overhead.

System call analysis was continued in the work of Simonsson *et al.* [25]. This paper presents ChaosOrca, a tool that performs CE experiments to assess the resilience of any docker-based microservice. This tool aims at evaluating a given application's self-protection capability with respect to system call errors. It is composed of i) the monitor, which is responsible for capturing the behavior of the system in runtime; ii) the perturbator, which injects different types of faults with respect to system calls and iii) the orchestrator that controls the first two to carry out the experiment and generates reports. This architecture was tested on three applications TTorrent, Nginx and Bookinfo.

Several works seek to test the resilience of a system in different infrastructures and at different levels. This involves analyzing and testing the availability of the systems in question. Yet, when it comes to security issues, these investigations fall short, since, to completely secure a system, it is necessary to take into account not only the availability of the service but also its integrity and confidentiality. For this reason, SCE is born, inherits the characteristics of CE, and focuses on security controls through experiments to ensure the capabilities of a system against attacks. However, since it is an emerging topic, there are very few academic works related to it.

In this sense, the first open source framework that demonstrated the value of applying CE to information security [24] was *ChaosSlingr* [3]. This tool was led 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[4].

The second framework that focuses on this field is presented in [28] as *CloudStrike*. It is defined as a software tool that applies Risk-Driven Fault Injection (RDFI) to cloud infrastructures. It is worth mentioning that the first version of this work is presented at [27]. Specifically, RDFI extends the application of CE to include cloud security by injecting security faults using attack graphs. The SCE-based proposal is then tested in some cloud services of leading platforms, namely, AWS and Google Cloud

[3] https://github.com/Optum/ChaoSlingr
[4] https://www.verica.io/

Platform. Interestingly, the authors claim they compute the risk to which the assets are exposed using the CVSS. Later on, the same authors leveraged the SCE strategies to test another proposal, CSBAuditor, a cloud security framework that can constantly monitor a specific cloud infrastructure to detect possible malicious activities [26].

7 Chaos Engineering Fundamentals

7.1 Key Concepts

The concept of CE was created 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 operation. For such reason, *ChaosMonkey* [4] was created with the aim of testing Netflix stability by injecting faults that randomly terminate internal instances. A year after launching *ChaosMonkey*, Netflix added new modes that report different types of faults or detected abnormal conditions. Each of these modes was considered a new simian, and together they formed what is known as *SimianArmy* [5].

In 2016 Kolton Andrus and Matthew Fornaciari founded Gremlin[5], 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" [13].

The CE experiments are based on the scientific method and should follow the CE principles [13] that define certain steps to guarantee that they are correctly executed. First, it is essential to mark out the normal behavior of the system. This process is known as defining the steady-state. Then, it is necessary to define a hypothesis that will be proved or refuted at the end of the experiment. Once the steady-state and hypothesis are set, experiments may be conducted introducing real-world events related to the hypothesis being proved, like instances that expose malfunctions, interrupted network connections, among others. Along with the execution of the experiments, metrics appropriated for the steady-state should be gathered so the hypothesis may

[5] https://www.gremlin.com/

be refuted or confirmed.

Although the principles of CE do not specify explicitly a observability component, it may be applied for the definition of experiments, since, for an experiment to be properly executed, observability helps to detect the normality of the system and even supports the variation of the steady-state [16]. Therefore, it is possible to affirm that CE and observability complement each other, and apply it in experiments can be done using traditional graphics (line charts, histograms, pie charts) or also visual metaphors that are considered a strategy to map from concepts and objects of an application domain to a system of similarities and analogies [23].

In addition to defining the hypothesis, steady state, and observability of each experiment, it is important to consider the Blast radius for each execution to ensure that the fallout from the experiment is minimized and contained[13]. In the same way, each experiment must have defined the set of actions to be carried out and the necessary probes to evaluate whether the hypothesis is disproved or not.

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, and the results that arise from conducting a CE experiment can help anticipate incidents, improve understanding of system failure modes and reduce maintenance costs [8].

Finally, to complement and justify all this theoretical component, as part of the learning process behind this thesis, the following courses were conducted: *Gremlin Certified Chaos Engineering Practitioner* [6] and *Gremlin Certified Chaos Engineering Professional*[7]. These courses allowed building the theoretical component of this work through the learning of topics such as: Facilitating CE practice with *Gremlin*; explaining CE attacks and their use cases; identifying the business and technical initiatives associated with each type of attack; facilitating CE experiments with *Gremlin*

and understanding what are the *GameDays* were evaluated. Both courses were successfully completed and certified, for the first course the certification can be found here and the second here.

7.2 Chaos Engineering Tools

A few CE frameworks may be found in-the-wild. It should be noted that the tools mentioned in Section 6 are tools that come from academia and are described in papers, books or magazines. However, the tools shown below are the most commonly used in the industry starting with the above-mentioned Gremlin, which allows one to experiment with more than ten different attack modes on different infrastructures. Nevertheless, not all of these modes are free, and it does not have reporting capabilities. Another known framework is ChaosMesh[6], which is an open-source cloud-native tool built on Kubernetes CRD (Custom Resource Definition). It allows testing several scenarios checking for network latency, system time manipulation, resource utilization, and more. Nonetheless, this tool does not have the advantage of scheduling attacks.

Another open-source CE framework is Litmus[7] 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)[8], another open-source tool that allows to automate and customize CE experiments by defining a set of probes and actions that may be pointed to different types of targets.

It should be noted that these well-known frameworks are not the only solution for exploring CE. They automate the process. However, there are command-line tools that allow performing several tasks that these frameworks already do, such as analyzing the use of resources in a system and detecting failures or malfunctions.

For example, for Linux users, there are several pre-installed comments that help to monitor different resources on the system. Table 1 shows some of these commands

[6] https://chaos-mesh.org/
[7] https://litmuschaos.io/
[8] https://chaostoolkit.org/

along with a brief description. Additionally, one can highlight the *stress*[9] tool that allows to impose loads and stress on the system. This tool can be used in conjunction with those shown above to observe and monitor the behavior of the system under load.

Tool	Resource	Description	Manuals
iostat	Disk	Looks in terms of throughput disk performance and utilization	Manual
sar	Network	Collects, reports and saves system metrics	Manual
free	RAM	Displays amount of free and used memory in the system	Manual
stress	CPU	Tool to impose load on and stress test systems	Manual

Table 1: Linux tools for monitoring resources

On the other hand, the BPF Compiler Collection (BCC)[10] is a set of tools for kernel tracing. Table 2 shows some tools for monitoring resources in the system. The added value of these tools is that they allow seeing the results in a more dynamic and user-friendly way.

Tool	Resource	Description	Example
biotop	Disk	Reads and identifies the load written to the disk.	Use Example
tcptop	Network	Summarize TCP send/recv throughput by host.	Use Example
oomkill	RAM	Traces the kernel out-of-memory killer, and prints basic details including the system load averages.	Use Example
cpudist	CPU	Measures the time a task spends on or off the CPU, and shows this time as a histogram.	Use Example

Table 2: BCC tools for kernerl tracing

8 Chaos Engineering experiments

Once the method and benefits of implementing CE are known, defining and implementing an experiment can be very simple. Several CE experiments are presented

[9] https://linux.die.net/man/1/stress
[10] https://github.com/iovisor/bcc

in [20]. These experiments aim to test the resilience of the systems in different environments such as containers (Docker[11] or kubernetes[12]) or the Java Virtual Machine (JVM) that allows to execute the compiled code of a program written in Java. Then, for a better understanding of the methodology applied to the CE experiments, four experiments were implemented and documented in this section: Experiment 1: Redis Latency, Experiment 2: Failed Requests on Redis, Experiment 3: Latency in *pgweb* and Experiment 4: Adding Failures to *pgweb*.

8.1 Inject faults at application level

The following two experiments consider the design of a product recommendation system for e-commerce customers. This recommendation was based on previous searches made by the user. In order to generate recommendations, a cookie was used to store the user's *sessionID* to provide personalized recommendations. In addition, to avoid latency problems, since this would lead to the clients' desertion, Redis was used. Redis is known as a in-memory data structure store used as a database, cache, message broker, and streaming engine[13]. It worked as a cache that stored the last three searches of each client.

Figure 1 presents the flow of the system. First, the user interacts with the interface by searching for an item. Then, the interface creates the *sessionID* and sends it to the search engine along with the item provided by the user so that the Redis client queries if that session already exists, if not, then stores it.

Once stored, the search engine queries the Redis client to request a user's interests, Redis stores the current interests and returns the user's last three searches. These last three searches are then analyzed with the *recommend()* method to finally provide an appropriate recommendation for the user based on their searches.

[11] https://www.docker.com/
[12] https://kubernetes.io/
[13] https://redis.io/

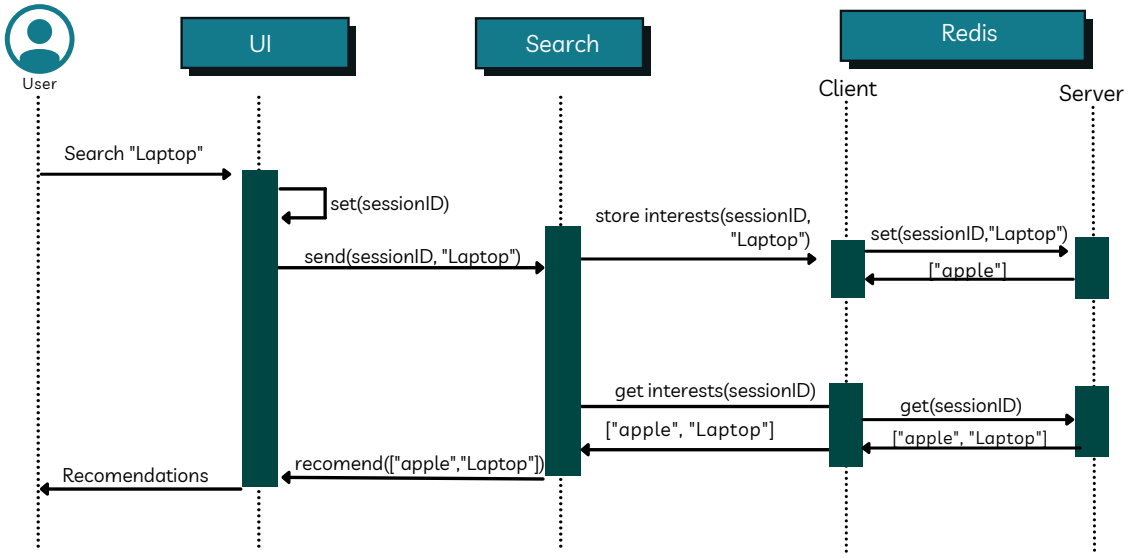


Figure 1: Execution flow for Experiments 1 and 2

8.1.1 Experiment 1: Redis Latency

This first experiment assumes that if the system slows down, users will start accessing competing sites. This is not a desirable case as it can result in financial losses for the business. Therefore, it is important to understand how the latency arising from the communication with Redis could affect the overall speed of the website.

As shown in Figure 1, the application is based on a Redis client which is accessed by the *get_interests()* and *store_interests()* functions, which in turn make use of the *get()* and *set()* methods to use the information stored in the cache. What this experiment does is to replace the original client with a chaos client which will inject latency in the *get()* and *set()* methods. This client will be used only when the environment variable "CHAOS" is activated when running the application.

With this, the following experiment is proposed following the CE methodology for its implementation

- Observability: Generate traffic and observe latency.
- Steady State: Observe latency without chaos changes.
- Hypothesis: If we add latency of 100ms to each interaction with the cache (writes and reads), then the overall latency of the search page should increase by 200ms.

Apache Bench was used to define observability and steady-state. This tool allows the generation of HTTP requests. In this case, POST requests were used to simulate a user's behavior searching the website. In this way, the total of completed and failed requests can be observed and the time taken for each of them.

Then, the idea of the experiment was to make the first scan during 10 seconds using Apache Bench in the normal state of the application, then activate the chaotic environment and run Apache Bench again for 10 seconds to observe the changes. The results shown by Apache Bench were summarized in Tables 3a and 3b for better understanding.

In Table 3a shows that a total of 859 complete requests were completed. Also, 85.7 requests per second were obtained for the steady state and the time taken for each request was on average 11.668 seconds. However, when adding latency to the application in a chaotic environment, Table 3b shows a total of 47 complete requests, noting that only 4.61 requests per second were sent and each one took an average time of 216.916 seconds.

Description	Metric
Complete Requests	859
Requests per Second	85.70 [sec]
Time per Request	11.668 [ms]

(a) Steady State Results

Description	Metric
Complete Requests	47
Requests per Second	4.61 [sec]
Time per Request	216.916 [ms]

(b) Execution Results

Table 3: Results for Redis Latency

With this experiment it was found that the hypothesis is confirmed because if latency of 100ms was added to each interaction with the cache, then the overall latency of the search page increases 200ms. Regarding the implementation, it is worth mentioning that adding chaos directly to the source code of the application can be a double-edged sword. It is simple to do but increases the chance of bugs being generated. Finally, the experiment emulated an unexpected behavior that allowed testing how the application components were affected and how the whole system reacted to this case.

8.1.2 Experiment 2: Failed Requests on Redis

Besides analyzing the latency in the application it is also fair to analyze the behavior in case of failure of any component. To prove the handling of exceptions, unit tests are usually performed. However, these do not allow testing the whole system. This is where CE comes into play since if this discipline is used as a kind of end-to-end test, the consequences of poor error handling on the client-side can be observed.

The experiment uses CE to check what would happen if the communication between i) the process (Redis Client) requesting to store the queries and ii) the cache (Redis server) that effectively stores them fails. In this case, the CE experiment is defined as follows:

- Observability: Navigate the application and view 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.

In this case, the steady-state of the experiment was given by the interaction with the website. The application showed its products and recommendations without any alteration. At the time the experiment was started, it was possible to observe that in some requests the application failed, however, the recommendation system handled the error and managed to recover automatically as soon as the access to the storage system was reestablished. Thus, it proves that the recommendation system is resilient to failures in the storage system.

8.2 Inject faults into a browser

For this section, it is assumed that a team is looking for an optimal way to manage their PostgreSQL databases. The team evaluated several open source options and decided to use *pgweb*, a UI for PostgreSQL databases. Pgweb allows to browse and export data, run searches and insert new information.

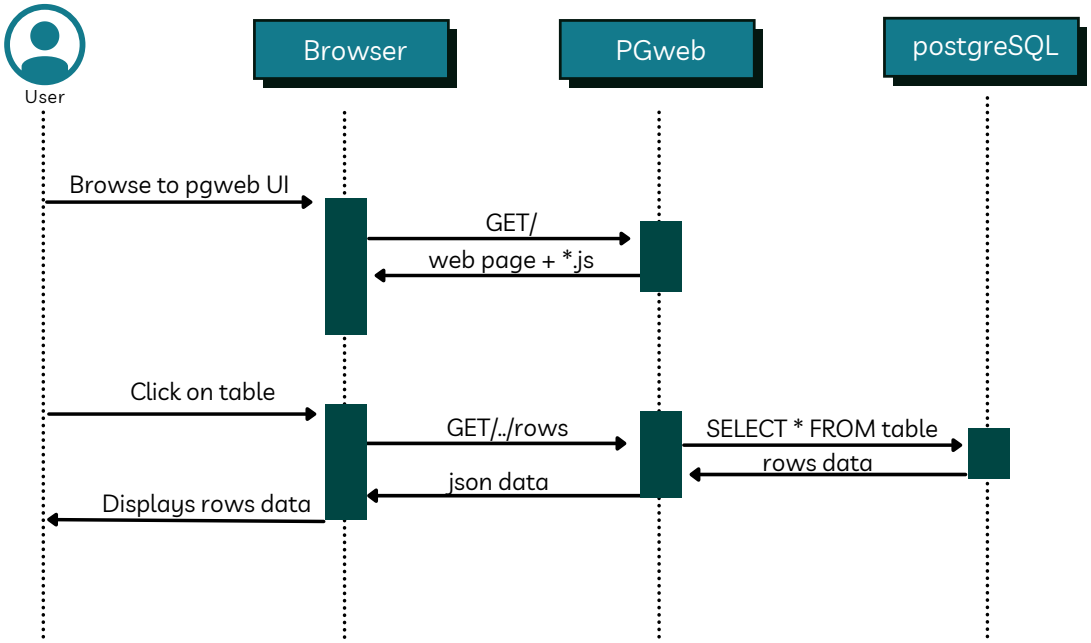


Figure 2: Execution flow for Experiments 3 and 4

Figure 2 shows the flow that *pgweb* follows to display the stored data. First, the user navigates the *pgweb* interface, then *pgweb* connects to the database and returns the web page along with the JavaScript code used to create it.

When the user interacts with one of the tables, the browser performs an HTTP GET request which makes a query to select the data in the selected table. To display the information to the user, the HTTP server reads the information returned by PostgreSQL in the form of a JSON and sends it to the user interface for display.

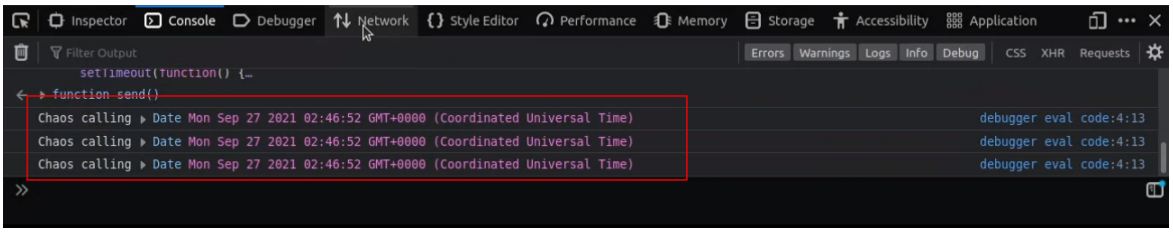
8.2.1 Experiment 3: Latency in *pgweb*

This experiment sought to analyze how latency would affect the behavior of pgweb. To achieve this, latency of 1 second was added to all requests generated by the code when selecting a table, and the response time of each request was checked. The experiment was defined as:

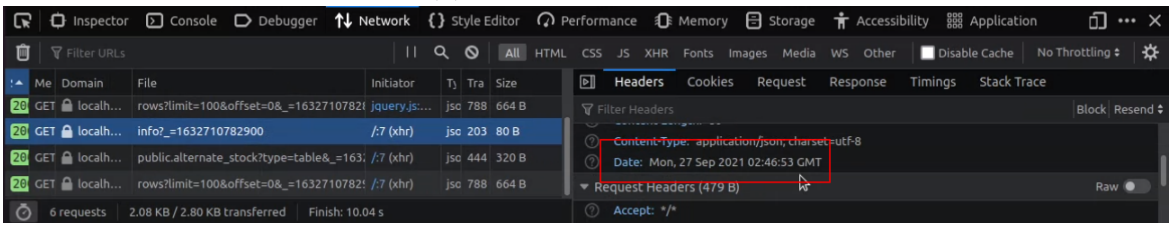
- Observability: The built-in Firefox timer is used to read the time it takes to execute the three requests made by the JavaScript code.
- Steady State: Browser measurements were taken before implementing the experiment.

- Hypothesis: If a latency of 1 second is added to all requests made from the application's JavaScript code, then the total time it takes to display the new table increases by 1 second.

To run the experiment, the method that sent the HTTP request has been modified by adding a timeout of 1 second before sending the request. When running the experiment, it was found that three requests were sent to the database at the moment of selecting a table, and it was observed that they did not depend on each other. Meaning that they were made in parallel, which allows concluding that the aggregate latency did not correspond to each request but to the set of the 3 requests.



(a) Chaos Calling Result



(b) Response time first request

Figure 3: Results for pgweb Latency

As can be seen in Figure 3a, the modified method printed on the console the date on which the requests were sent. In this case, it can be seen how all of them are sent at the same time. Nonetheless, Figure 3b shows that the response time of the selected request was one second after the message was printed in the console. This allows confirming the hypothesis since there is a one-second difference but not between each request.

8.2.2 Experiment 4: Adding Failures to *pgweb*

This last experiment starts from the idea that when launching *pgweb* locally, connectivity problems are not experienced, but in real life, it is very likely to happen. Therefore,

for this experiment, the goal is to analyze how the application should behave when there are connectivity problems. Normally, a user would try to retry the request and if it fails again, then the application should display an error message to avoid displaying obsolete or inconsistent data. Thus, the following experiment structure is defined:

- Observability: Observe if the user interface shows errors or obsolete data when selecting a table.
- Steady State: There are no errors or obsolete data.
- Hypothesis: If an error is added in some requests made by the JavaScript interface, then an error message and no inconsistent data should be displayed every time a table is selected.

In this case, an error message has been injected once or twice when sending the HTTP request. And the experiment consisted of browsing the application and analyzing whether an error message was provided to the user every time the requests threw the error.

However, when running the experiment and navigating the application, it was observed that the tables were not always refreshed when they were selected. That is, there was inconsistency in the data displayed. Also, the application did not show any error message associated with what happened to notify the user that something unusual was occurring. Therefore, it can be concluded that the application is not resilient to errors and therefore, the hypothesis was refuted.

9 Towards Security Chaos Engineering

9.1 Key Concepts

Using CE, testing security in systems under the premise that "failure is the greatest teacher" is possible. This idea was first proposed by Aaron Rinehart, who pursued the application of CE in cybersecurity while working as Chief Security Architecture at

UnitedHealth Group. As mentioned in the previous section, CE has been 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 availability but also other attributes such as integrity and confidentiality to boost the concept of SCE. This concept, 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" [22].

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, bad configurations, logic flaws, and insecure design, among others. In addition, SCE may help in the reduction of security incidents and remediation costs, if SCE experiments are executed frequently, as it allows developers to: i) understand their system, ii) define a response plan, iii) identify system modules failing or iv) note that some components were omitted during development. Also, SCE minimizes impacts on users through experimentation, which in turn improves the ability of developers to track and measure security.

The main factors motivating the implementation of SCE can be seen in Figure 4. First of all, SCE allows finding emergent properties in a system, either unexpected behaviors or properties that arise from the interaction with components that flow differently from the usual operation of the application. Secondly, it allows minimizing risks such as data leakage by testing confidentiality, data corruption compromising data integrity, and even component failures affecting availability. Finally, SCE allows testing of the entire system by integrating different types of tests such as unit testing, end to end, and integration testing.

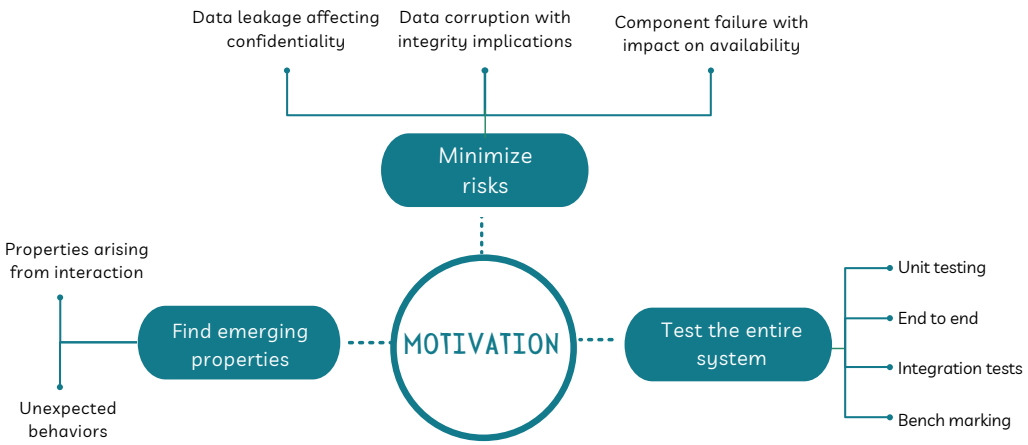


Figure 4: Motivation for SCE

9.2 Security Chaos Engineering Tools

Very few SCE frameworks exist today, with ChaoSlingr [22] and CloudStrike [28] mentioned in the state of the art (Section 6) being two good examples.

Using this fact as motivation to generate a new framework for SCE, an analysis of each of the most common vulnerabilities in web applications compiled in the OWASP TOP 10 and the different experiments that can be done with CE was performed. As a result of this review, the matrix shown in Table 4 was constructed.

The rows show each of the vulnerabilities mentioned in the OWASP Top 10 for 2021. In the case of the columns, the most basic and common CE experiments were considered. The following is a brief description of each of them.

- **Blackhole**: These experiments test the availability of the network by testing its response when system dependencies are not available. In other words, a blackhole experiment simulates the unavailability of dependencies by dropping network traffic between services.

- **Latency**: The idea for these experiments is to inject a delay into network traffic to validate system resilience to slow network scenarios.

- **Error Injection**: These are used to test the error handling of an application.

✓ Applicable ✗ Not applicable ≈ Partially applicable	Blackhole	Latency	Error Injection	Disk Usage	Packet loss
Broken Access Control	✗	✗	✗	✗	✗
Cryptographic Failures	✗	✗	✗	≈	✗
Injection	✗	✗	✓	✗	✗
Insecure Design	≈	≈	≈	≈	≈
Security Misconfiguration	✓	✓	✓	✗	✗
Vulnerable and Outdated Components	✓	✗	✓	✗	✗
Identification and Authentication Failures	✗	✓	✗	✓	✗
Software and Data Integrity Failures	✗	✓	✗	✗	✓
Security Logging and Monitoring Failures	✓	✗	✓	✓	✗
Server-Side Request Forgery (SSRF)	✗	≈	≈	≈	✗

Table 4: CE Experiments Vs Vulnerabilities

Exceptions that may occur in production are launched and the response of the application is evaluated.

- **Disk Usage**: These experiments are created to simulate disk behavior against reads and writes of a very large data set to test disk capacity for situations such as data migration or recovery jobs.

- **Packet Loss**: These experiments simulate the state of a congested network where packets may be dropped or lost. this allows for testing the user experience when a percentage of packets are lost or corrupted.

To define each intersection, an assessment was performed on the application layers where vulnerabilities can be found, and then the CE experiment was identified to test that layer. For example, code injection vulnerabilities can be directly associated with error injection experiments because errors can be thrown either from the database (testing SQL injection) or from the system (testing command injection) and evaluate if the application handles the error correctly or if it is otherwise vulnerable to code injections.

In Table 4, it is also feasible to identify vulnerabilities where the relationship is

not so direct and therefore the experiments are defined as partially applicable. This is because the experiments must be adapted and modified for a specific use case. For example, the insecure design vulnerability is very broad and depends very much on the software that is being considered for implementation.

Finally, the cells that indicate that an experiment is not applicable were defined by the scope of the experiment, since there was probably no direct relationship with the layer in which the vulnerability can be generated, or because proving its existence requires a very high manual component, which is not easy to achieve with these specific experiments. For example, to exploit a Broken Access Control, the attacker must go through different attack techniques such as user enumeration processes, the use of rainbow tables to access credentials, or social engineering processes such as phishing, and this is not easy to achieve with the mapped experiments.

This matrix has been the first approach from this degree work to the integration of security in CE. However, the nature of these experiments evaluates only the availability of a service or system, but security goes beyond that, as it is necessary to maintain integrity and confidentiality as well.

9.3 Benefits of SCE

SCE has been identified as a source of knowledge that can contribute significantly to the field of computer security. Specifically, it is able to support ethical hacking and pentesting exercises.

The ethical hacking and pentesting processes allow attacking different targets by finding and exploiting vulnerabilities. Specifically, the pentensting process groups a set of offensive activities (manual and automated) performed by a red team, which at the same time are contained through defensive activities conducted by a blue team. In most cases, pentesting exercises involve time and effort that will result in an extensive report with the characterization of the vulnerabilities found and exploited, and generally a set of remediation proposals. In organizations with an intermediate maturity security level, pentesting exercises are developed regularly, for example, every 3 or 6

months.

In this context, SCE may improve a traditional pententing process as it offers an alternative way of detecting vulnerabilities in targets, providing a new tactic that enriches the existing tool-set of blue and red teams. SCE experiments can be performed both in production and testing environments, in contrast with the pentesting, whose main focus is production ones. This allows testing the system in the early stages of development and reduces remediation costs. Additionally, when considering complex or distributed systems, SCE experiments help to understand the system as a whole, going beyond unit tests over specific components. It allows not only to test for system errors but also for assumptions about the system, such as component configuration or human errors.

Moreover, pentesting is focused on the use of tools with an offensive purpose and is generally executed by personnel external to the organization, while SCE is focused on tools that seek to build a more defensive strategy and are generally executed by the organization's internal personnel. Finally, pentesting techniques are generally manual and there is no specific method to follow. In contrast, SCE experiment execution procedures have a very high automation component and follow CE principles, which are aligned with the scientific method.

10 Proposal of ChaosXploit

ChaosXploit[14] is a SCE-powered framework composed of different modules that support the application of CE methodology to test security in different kinds of information systems. The architecture of the proposal is depicted in Figure 5, and each internal module is described in the following sections.

[14] https://github.com/SaraPalaciosCh/ChaosXploit

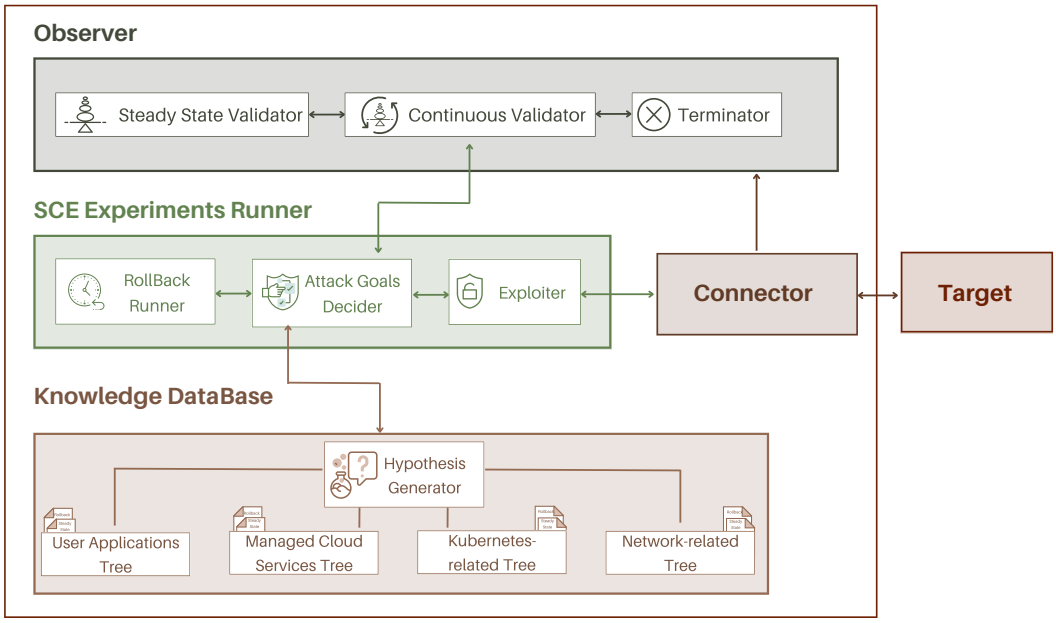


Figure 5: Proposed architecture of ChaosXploit

10.1 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 mature a defensive strategy. Thus, this module is composed of a set of attack trees and a hypothesis generator.

- **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, a HTTP request or some process to be run on the operating system. Also, each attack tree contains a set of steady states and rollbacks associated with the actions defined in the tree to support the execution of the experiment. 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.

- **Hypothesis Generator**: Intelligence contained in the attack trees needs to be converted to a hypothesis so 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[15] is pointed out to the main domain.

10.2 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 to monitor specific conditions of the target before, along and after the execution of the SCE experiments. This module is composed of a steady state validator, a continuous validator and a terminator.

- **Steady state Validator**: The steady state validator is in charge of verifying the steady state hypothesis in the target that represents the conditions that are considered normal. Normal conditions will depend on the attack goal and the specific hypothesis being tested, for example a normal condition may be a well-formed response from a web server.

- **Continuous validator**: The continuous validator permits verifying specific signals detected from the target, which allows determining the results of an interaction between the exploiter and the target. These signals are particularly important because they may indicate if a current action included in a branch of an attack tree was successful, so the following action in the branch should be triggered, or they simply may indicate that target is not vulnerable and following

[15] https://github.com/ElevenPaths/FOCA

actions of the branch should not be executed.

- **Terminator**: The terminator observes the failure states of the SCE experiment to define the actions to follow consequently. for example, if the target gets unresponsive due to the execution of a SCE experiment, a failure state will be launched and the terminator will able to inform to the Rollback Runner so it can restore the target.

10.3 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 leads the interaction with the target but it also centralizes the communication with the observer and knowledge database. It consists of three main elements: attack goal decider, exploiter, and rollback runner.

- **Attack goal decider**: The attack goal decider receives a defined goal attack as input to be tested over a target. Such 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.

- **Exploiter**: The exploiter executes the SCE experiment over a target with the purpose of validating or refuting a hypothesis. 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.

- **Rollback runner**: An experiment may contain a sequence of actions that reverse what was undone during the experiment. These 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.

10.4 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.

The main essence of this architecture is that it allows the automation of CE experiments. Although this is a generic infrastructure, it should be noted that the effectiveness of automation of each experiment depends on the attack goal defined in the tree associated with the experiment since it is possible that not all branches can be easily automated. However, the human role could play a significant part in supporting the execution process, validating those phases that are not easily automated.

10.5 Flow Diagram

The interactions between the components of ChaosXploit are shown in Figure 6.

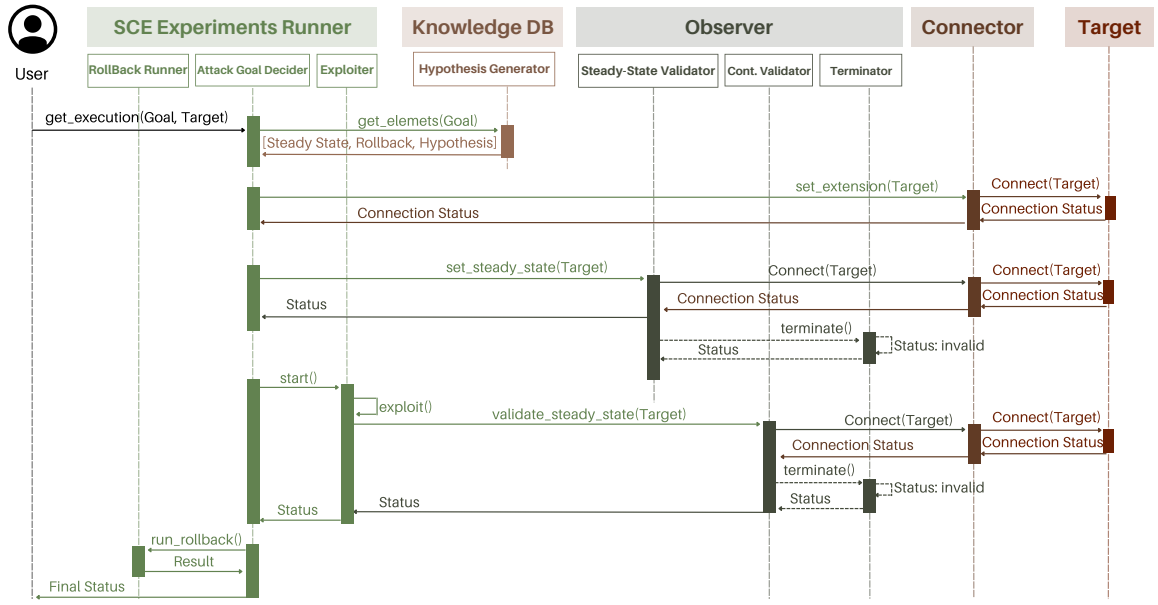


Figure 6: Flow diagram of the execution of a SCE experiment in ChaosXploit

First, the user of ChaosXploit request to the Attack Goal Decider 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 gets from

the knowledge database the steady state of the experiment, the rollback procedure, and the most proper hypothesis (attack tree) that matches the attack goal desired by the user. The Attack Goal Decider also request to the Connector the preparation of the proper extension for the target informed by the user. When a connection to the target is established and a hypothesis is defined, then the Attack Goal Decider does the following actions: i) set the steady state of the experiment in the Observer, ii) start the execution of the steps defined in the first branch of the attack tree with the help of the Exploiter, and iii) keep 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 was achieved. If the Continuous Validation fails, the termination process is activated with the Terminator. The experiment ends with the execution of the Rollback Runner to restore everything.

11 Experiments

Multiple experiments have been conducted using the ChaosXploit proposal mentioned earlier. Based on the fact that AWS S3 buckets and Elasticsearch databases account for nearly 45% of the cloud misconfigured and compromised technologies [21], for this experiment, ChaosXploit focuses on evaluating the security of the AWS S3 service considering the possible configurations 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 is composed of the following subsections: Settings 11.1, where the hardware and software requirements to carry out the experiment, are specified. Definition of the knowledge database 11.2, where the attack tree is presented together with the specification of the branch chosen for the experiment. SCE experiment 11.3 in which the steady state and the hypothesis of the experiment are defined, as well as the input parameters and the monitored variables.

11.1 Settings

The following setup was used to make use of ChaosXploit:

- Hardware: the experiment was executed on a Fedora OS with AMD Ryzen 5 3500U CPU, 8GB RAM and 512GB SSD.
- Internal Components: Some of the components of ChaosXploit have been built over existing modules of ChaosToolkit, as it is a open source framework that allow its extension and improvement to make it oriented to security purposes. ChaosToolkit was chosen since this tool allows to automate the experiments in a simple way 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.

11.2 Definition of the Knowledge Database

In Figure 7 it is possible to observe the attack tree implemented for this experiment.

It starts with the attacker finding public buckets by either enumerating the names or searching sites such as the Wayback Machine. Then, the next action seeks to confirm if the attacker succeeds in establishing a connection to the bucket. Once the connection is established, the attacker can follow one of the 4 different branches to reach the attack goal identified in the tree as the last box: extract or modify information. These paths are described as:

- **Branch 1**: Where the attacker has gained access to the bucket without any permission or authentication process. Once inside, he can make an inspection of the objects contained in the storage system, and read the Access Control Lists (ACL). If these ACLs have permissions open to the entire public, then the attacker will be able to reach the attack goal.

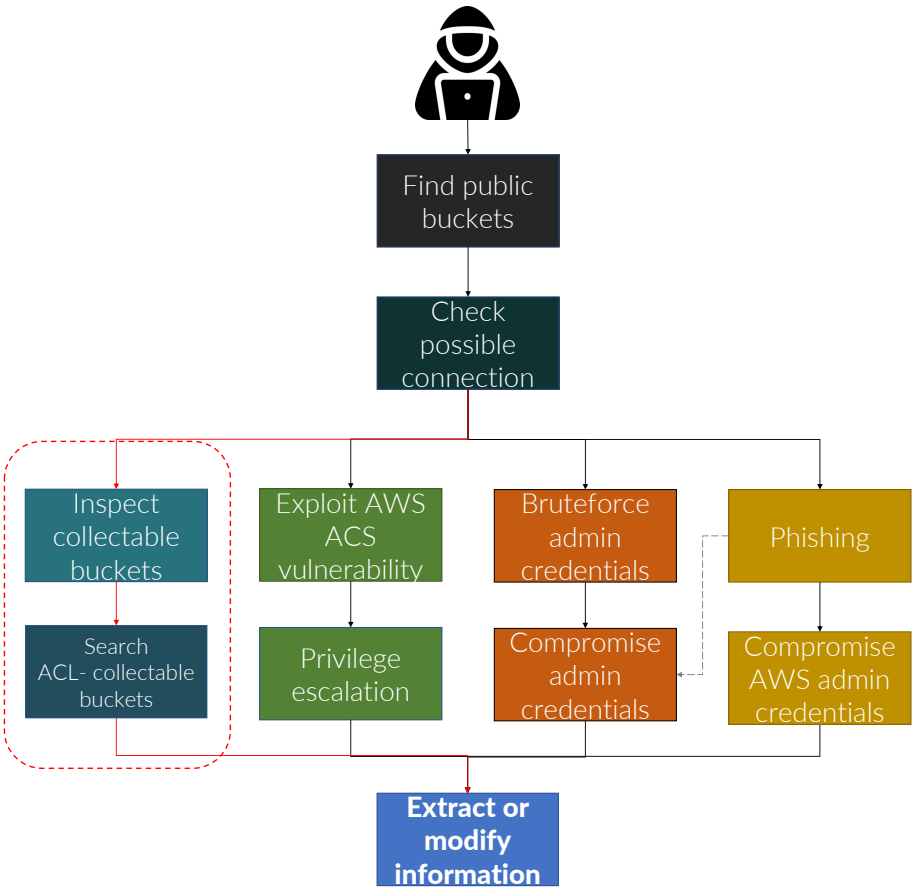


Figure 7: Attack Tree for the experimental scenario, highlighting the implemented path

- **Branch 2**: It is a path taken by the attacker in case the bucket has the access permissions properly configured. At this point, the attacker could make use of possible vulnerabilities in the AWS access control system, also known as IAM, to then elevate his privileges and gain access to the bucket's information, thus achieving the attack goal.

- **Branch 3**: In which the attacker can use of brute-forcing techniques to compromise admin credentials and thereby gain access.

- **Branch 4**: Where the attacker can use of social engineering techniques such as phishing to compromise credentials and gain access.

It is important to note that the execution of the first branch was included in the scope of this project, since the actions included in that branch were fully automatable.

Other branches could also be implemented through a combination of manual and automatic actions.

11.3 SCE experiment

The goal of this experiment stems from the fact that amazon s3 allows data to be stored and protected from unauthorized access with encryption features and access management tools. However, the shared responsibility mode of cloud services has led the creators of this type of storage to commit flaws during security configuration. Leaving the information open to the public, putting its confidentiality, integrity, and availability at risk.

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

- Observability: Public AWS S3 Buckets.
- 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.

Implementation of the first branch of the attack tree defined for this scenario is described below. First the finding of public buckets was done using enumeration techniques by considering regular expressions. Since amazon s3 has defined a series of requirements for the bucket names, this makes it very easy for the attacker to enumerate them. Then, the connection check was performed using boto3, the AWS SDK for python. With this step, the buckets were cleaned, leaving out those that no longer exist or had invalid names. Afterward, ChaosXploit inspects the buckets to identify if their objects can be read and finally searches if there are buckets that allow access to the ACLs.

As shown in Table 5, different parameters were considered as input values for ChaosXploit.

Monitored Variables	
Name	**Description**
Object - Collectable	No. of buckets that have public objects and are accessible by anyone
ACL-Collectable	No. of buckets that have public ACLs and are accessible by anyone
Permissions	No. of permissions obtained from the ACL.
Input Parameters	
Name	**Description**
Domain(Optional)	Domain name to which you want to identify the buckets
Threads	Execution Threads
Mode	Object-Collectable or ACL-Collectable
Output	Output File Name

Table 5: Monitored variables and input parameters for experiments.

First, the *domain* is an optional input that should contain the name of the organization to be analyzed. This option was considered 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 bruteforce, 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 buckets 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* or *ACL-Collectable* buckets. The last input, *output*, is a file name used to store the results and feed the ChaosXploit continuous validator.

Regarding the monitored variables, three were considered: i) Buckets that have public objects that can be accessed by anyone, denoted by **Object-Collectable** in Table 5, ii) Buckets that have public ACLs and can be accessed by anyone denoted by **ACL-Collectable** and iii) the **Permissions** obtained from the ACLs.

11.4 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 tools such as s3enum[16], bucketkicker[17] or Sublist3r[18].

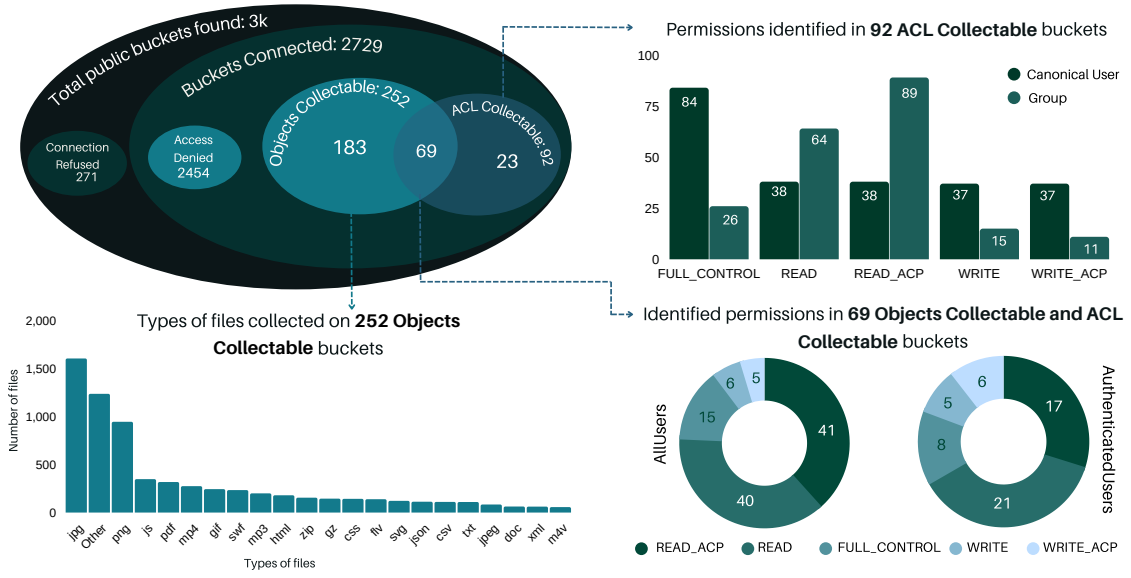


Figure 8: Results of the execution of each action included in the first branch of the attack tree

As seen in the upper left part of Figure 8, all possible actions of the attack tree were executed by ChaosXploit. It is possible to identify that for the second one (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, i.e. it did not follow the common bucket naming characteristics proposed by AWS. This leaves us with 2729 buckets remaining to test.

In the case of the third action of the attack tree (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.

[16] https://github.com/koenrh/s3enum
[17] https://github.com/craighays/bucketkicker
[18] https://github.com/aboul3la/Sublist3r

The lower left part of Figure 8 shows the file extensions that were extracted from 252 buckets that were Object Collectable. From each bucket only the first 50 objects were collected, since some buckets had more than 100000 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 first consider that Amazon S3 has a set of predefined groups:

- **Authenticated Users group:** Representing all AWS accounts.
- **All Users group:** Allowing anyone in the world to access the resource.
- **Log Delivery group:** allowing access logs to be written to the bucket.

Additionally, AWS defines also the following types of permissions:

- **READ:** Allows grantee to list the objects in the bucket.
- **WRITE:** allows grantee to create new objects in the bucket. For the bucket and object owners of existing objects, also allows deletions and overwrites of those objects.
- **READ_ACP:** Allows grantee to read the bucket ACL
- **WRITE_ACP:** Allows grantee to write the ACL for the applicable bucket.
- **FULL_CONTROL:** Allows grantee the READ, WRITE, READ ACP, and WRITE ACP permissions on the bucket

In the upper right part of Figure 8 is possible to identify that 92 of the 257 buckets allowed the extraction of the ACLs. Up to 13 permissions per bucket were identified. These showed information about the user who owned the bucket, known as **CanonicalUser** by AWS, or about the user groups that had access to it. Then, it is worth noting that for canonical users the FULL_CONTROL permission was enabled for 84

buckets (91.3%), and in the case of the user 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, the last step is to analyze the results of those buckets that allowed the extraction of both objects and ACLs. As seen in the lower right part of Figure 8, 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 *AuthentucadedUsers* group allowed the modification of their content (WRITE permission) and the alteration of the ACLs (WRITE_ACP permission), indicating a big flaw that could compromise severally the confidentiality, integrity and availability of the stored data.

With these results, it was noted 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. For example, for the case presented above, resilience strategies can be taken to secure the analyzed objects so that they do not fall back into misconfiguration. These strategies can range from the encryption of the stored information, the correct configuration of the Access Control Lists so that they cannot be modified, to the simple action of configuring the bucket as private to avoid information manipulation.

12 Results Summary

This section describes very briefly the results obtained throughout the development of the project. The first phase of this project was completely exploratory, so this phase provided 2 official certificates as practitioner and professional in CE granted by Gremlin to reinforce the concepts mentioned in Section 7 and the implementation and documentation of 4 CE experiments to understand the use of the CE methodology shown in Section 8.

The second phase focused on the proposal described in Section 10 of ChaosXploit as an SCE framework for vulnerability detection and mitigation support. Additionally, a set of ChaosXploit validation experiments on a real scenario was presented. It should be noted that all the development was published in a GitHub repository and is fully accessible.

Finally, as a result of this research, a summary article was submitted to the VI Cybersecurity Research Conference (JNIC, Bilbao, Spain) in the category "Cybersecurity Research".

13 Conclusions and Future Work

No one could expect the impactful digital revolution we live in, changing substantially how we live our lives with great benefit. 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 services. Such threat entities use increasingly advanced techniques, for example based on malware campaigns [14] or threats addressed to a specific technology [17].

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

This degree work enabled to perform an exploratory research in terms of the analysis that was developed during its first phase. This component allowed the understanding of the essence of chaos engineering and its wide variety of applications. As well as the collection and implementation of experiments based on chaos engineering to understand its methodology and the value provided by its application in different fields.

Thanks to this learning process carried out in the first semester of this project, an

opportunity to contribute to the field of cybersecurity by integrating chaos engineering was identified. This integration is known as security chaos engineering and was the driving force behind the creation of ChaosXploit.

ChaosXploit helped to work on the experimental axis of this research, also defined as the second phase, as it is a SCE-powered framework that is able to conduct Security Chaos Engineering 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. Also, ChaosXploit features an observer, which in turn 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. Results demonstrated that the approach can be successful, highlighting several unprotected buckets for a specific attack path.

It is important to mention that the work with ChaosXploit can be fully extended. The point where this research ends opens the possibility of extending the target architectures of the ChaosXploit framework in future work to include other use cases, systems, or vendors. In addition, the integration of a recommendation module that suggests countermeasures once a security flaw is discovered is worth investigating. On the other hand, the performance of ChaosXploit should be further evaluated to definitively demonstrate its usefulness in performance-demanding scenarios.

References

[1] H. Adkins, B. Beyer, P. Blankinship, A. Oprea, P. Lewandowski, and A. Stubblefield. *Building Secure and Reliable Systems: Best Practices for Designing, Implementing, and Maintaining Systems*. O'Reilly Media, 2020.

[2] Ali Basiri, Lorin Hochstein, Nora Jones, and Haley Tucker. Automating chaos experiments in production. *CoRR*, abs/1905.04648, 2019.

[3] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, Inc., 1st edition, 2016.

[4] Netflix Technology Blog. Netflix chaos monkey upgraded. https://netflixtechblog.com/netflix-chaos-monkey-upgraded-1d679429be5d. Last time accessed: 2022-03-14.

[5] Netflix Technology Blog. The netflix simian army. https://netflixtechblog.com/the-netflix-simian-army-16e57fbab116. Last time accessed: 2022-03-14.

[6] Tammy Butow. Gremlin certified chaos engineering practiotioner. Practitioner. Last time Accessed: 2021-11-10.

[7] Tammy Butow. Gremlin certified chaos engineering professional. Professional. Last time Accessed: 2021-11-10.

[8] Tammy Buttow. Chaos engineering: the history, principles, and practice. https://www.gremlin.com/community/tutorials/chaos-engineering-the-history-principles-and-practice/. Last time accessed: 2022-03-21.

[9] Carlos Camacho, Pablo C. Cañizares, Luis Llana, and Alberto Núñez. Chaos as a Software Product Line—A platform for improving open hybrid-cloud systems resiliency. *Software - Practice and Experience*, pages 1–34, 2022.

[10] Daniel Díaz-López, María Blanco Uribe, Claudia Santiago Cely, Daniel Tarquino Murgueitio, Edwin Garcia Garcia, Pantaleone Nespoli, and Félix Gómez Mármol. Developing secure iot services: A security-oriented review of iot platforms. *Symmetry*, 10(12), 2018.

[11] Daniel Díaz-López, Ginés Dólera Tormo, Félix Gómez Mármol, Jose M. Alcaraz Calero, and Gregorio Martínez Pérez. Live digital, remember digital: State of the

art and research challenges. *Computers & Electrical Engineering*, 40(1):109–120, 2014. 40th-year commemorative issue.

[12] Alya Hannah Ahmad Kamal, Caryn Chuah Yi Yen, Gan Jia Hui, Pang Sze Ling, and Fatima tuz Zahra. Risk assessment, threat modeling and security testing in sdlc, 2020.

[13] Mathias Lafeldt and Gu Yu. Principles of chaos engineering. https://principlesofchaos.org/. Last time accessed: 2021-11-16.

[14] Isabella Martínez Martínez, Andrés Florián Quitián, Daniel Díaz-López, Pantaleone Nespoli, and Félix Gómez Mármol. Malseirs: Forecasting malware spread based on compartmental models in epidemiology. *Complexity*, 2021, 2021.

[15] Francesc Mateo Tudela, Juan-Ramón Bermejo Higuera, Javier Bermejo Higuera, Juan-Antonio Sicilia Montalvo, and Michael I. Argyros. On combining static, dynamic and interactive analysis security testing tools to improve owasp top ten security vulnerability detection in web applications. *Applied Sciences*, 10(24), 2020.

[16] R. Miles. *Chaos Engineering Observability*. O'Reilly Media, Incorporated, 2019.

[17] Pantaleone Nespoli, Daniel Díaz-López, and Félix Gómez Mármol. Cyberprotection in iot environments: A dynamic rule-based solution to defend smart devices. *Journal of Information Security and Applications*, 60:102878, 2021.

[18] OWASP. Owasp org. https://owasp.org/. Last time Accessed: 2021-05-14.

[19] OWASP. Owasp top ten. https://owasp.org/www-project-top-ten/. Last time Accessed: 2021-05-12.

[20] M. Pawlikowski. *Chaos Engineering: Site reliability through controlled disruption*. Manning, 2021.

[21] Rapid7. 2021 cloud misconfiguration report. Technical report, 2021.

[22] Aaron Rinehart and Kelly Shortridge. Security chaos engineering gaining confidence in resilience and safety at speed and scale. Technical report, 2020.

[23] Yury Niño Roa. Chaos engineering and observability with visual metaphors. 2022.

[24] C. Rosenthal and N. Jones. *Chaos Engineering: System Resiliency in Practice*. O'Reilly Media, 2020.

[25] Jesper Simonsson, Long Zhang, Brice Morin, Benoit Baudry, and Martin Monperrus. Observability and chaos engineering on system calls for containerized applications in Docker. *Future Generation Computer Systems*, 122:117–129, 2021.

[26] K. A. Torkura, Muhammad Sukmana, Feng Cheng, and Christoph Meinel. Continuous auditing and threat detection in multi-cloud infrastructure. *Computers and Security*, 102:102124, 2021.

[27] Kennedy A. Torkura, Muhammad I.H. Sukmana, Feng Cheng, and Christoph Meinel. Security chaos engineering for cloud services: Work in progress. In *2019 IEEE 18th International Symposium on Network Computing and Applications, NCA 2019*. Institute of Electrical and Electronics Engineers Inc., sep 2019.

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

[29] Long Zhang, Brice Morin, Benoit Baudry, and Martin Monperrus. Maximizing error injection realism for chaos engineering with system calls. *IEEE Transactions on Dependable and Secure Computing*, pages 1–1, 2021.

[30] Long Zhang, Brice Morin, Philipp Haller, Benoit Baudry, and Martin Monperrus. A Chaos Engineering System for Live Analysis and Falsification of Exception-Handling in the JVM. *IEEE Transactions on Software Engineering*, 47(11):2534–2548, 2018.