ChaosXploit: A Security Chaos Engineering framework based on Attack Trees

Sara Palacios Chavarro[1], Daniel Díaz-López[1], Pantaleone Nespoli[2]
[1]School of Engineering, Science and Technology, Universidad del Rosario, Bogotá, Colombia
{sara.palaciosc, danielo.diaz}@urosario.edu.co
[2]Department of Information and Communications Engineering, University of Murcia, 30100, Murcia, Spain
pantaleone.nespoli@um.es

***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. 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 to protect assets under cyber risk scenarios. This paper proposes ChaosXploit, a security chaos engineering framework based on attack trees, which leverages the chaos engineering 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 proposal 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.**

***Index Terms*—Security Chaos Engineering, attack trees, cloud managed services, vulnerabilities**

Contribution type: *Original research*

I. 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 [1]. Although SRE has been approached with different methodologies, over the last ten years, a new approach for testing the resiliency of distributed systems has emerged [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 subsequently mitigated. CE tests are designed to "build confidence in the system's capability to withstand turbulent conditions in production" [3].

Designing CE experiments implies defining a prepared and controlled environment to analyze a target system [4] and applying a scientific method that allows one 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, checking these elements may not be enough if the ultimate goal is a holistic validation of the system's security level. It might be the case in different distributed systems, such as secure IoT services [5] or personal data managers with high-security requirements [6].

Considering what was previously said, some efforts have come up towards applying CE to cybersecurity in the last five years, 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 a holistic cybersecurity perspective, i.e., confidentiality, integrity, and availability [7].

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

The main contributions of this paper are summarized as follows:

- The proposal of a SCE framework named ChaosXploit, which uses attack trees as the main flowchart for the execution of attacks, and contains three main components: an observer, an experiment runner, and a knowledge database.
- The design of an attack tree that pursues an attack goal of extraction or modification of information of AWS S3 buckets that enriches the knowledge database of ChaosXploit.
- 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.

This paper is structured as follows: Section II collects the most recent works proposed related to SCE, exploring their pros and cons. Then, Section III describes ChaosXploit, our proposed framework to conduct SCE experiments. Next, in Section IV some experiments over ChaosXploit are proposed are executed. Finally, Section V concludes the work, showing some future work that can improve our proposal.

II. State of the art

Several research works have been proposed in the literature so far that leverage the robust capabilities of CE. Nevertheless, the application of the methodology, together with its definition, has been ambiguous.

Since the release of *Chaos Monkey* in 2011 by Netflix [8], CE has been chiefly applied to test the resilience of cloud and virtualized infrastructures, arguing on the potential benefits that the chaotic methodology could bring.

In this sense, Camacho *et al.* [9] proposed *Pystol*, a fault injection platform to test the resiliency of hybrid-cloud systems in adverse circumstances. Available as an open-source framework, Pystol exploits CE's abilities in the form of a Software Product Line (SPL) that can be mounted on top of cloud ecosystems. The platform is then tested in a production-ready environment, executed using standard Kubernetes objects and APIs and Amazon Web Services to deploy the cluster with three use cases.

Furthermore, the work in [10] presented *ChaosOrca*, an open-source CE-based fault injector for system calls in containerized applications. That is, the system can estimate the self-protection capability of any Docker-based microservice concerning system call errors. In particular, ChaosOrca formalizes the steady-state of the container by automatically recording several system metrics (e.g., CPU and RAM consumption, network I/O). Then, perturbations are injected into the system calls invoked by the dockerized application in an isolated fashion, without impacting the normal operations of possible other containers. The prototype is tested in three case studies of Docker microservices, namely Torrent, Nginx, and Bookinfo, showing promising results in detecting resilience weaknesses.

Moreover, Zhang *et al.* [11] proposed *ChaosMachine*, an open-source and extensible CE system in Java aiming to analyze the exception-handling capabilities in production environments. So, ChaosMachine can reveal potential resilience problems of try-catch blocks with an architecture composed of three components: i) a monitoring sidecar, ii) a perturbation injector, and iii) the chaos controller. The framework is then tested with three large-scale open-source Java applications summing 630k code lines with realistic workloads, demonstrating its capacities in production environments.

Recently, the main target of CE has been slightly moved from resilience to including security concerns surrounding a system. Assuming that security failures are going to happen doubtless, SCE aims at testing the security controls of a system through proactive experiments and, thus, building confidence in the system's capabilities to defend against malicious conditions. Unfortunately, since this paradigm change has happened lately, the amount of academic work and tools are still scarce. To this extent, *ChaoSlingr* is the first open-source software tool to demonstrate the possibility of applying the principles of CE to information security[1]. The system was designed to operate on AWS by a team at UnitedHealth Group led by Aaron Rinehart to exhibit a simplified manner for writing security chaos experiments [12]. From the main project, several organizations have started to utilize ChaosSlingr to design their chaotic experiments.

Additionally, the work in [7] presented *CloudStrike*, a software tool that applies Risk-Driven Fault Injection (RDFI) to cloud infrastructures. For the sake of the reader, the tool was firstly proposed in [13]. Specifically, RDFI extends the application of CE to include cloud security without losing the resilience viewpoint, i.e., 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 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 [14].

Also, the application of SCE to enhance API security is defined in [15]. Particularly, the authors propose utilizing this methodology to test the configuration of the API's security controls, exposing early vulnerabilities.

III. Proposal of ChaosXploit

This section describes ChaosXploit, 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 1, and each internal module is described in the following sections.

A. 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.

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.

2) Hypothesis Generator: The 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[2] is pointed out to the main domain.

[1]https://github.com/Optum/ChaoSlingr

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

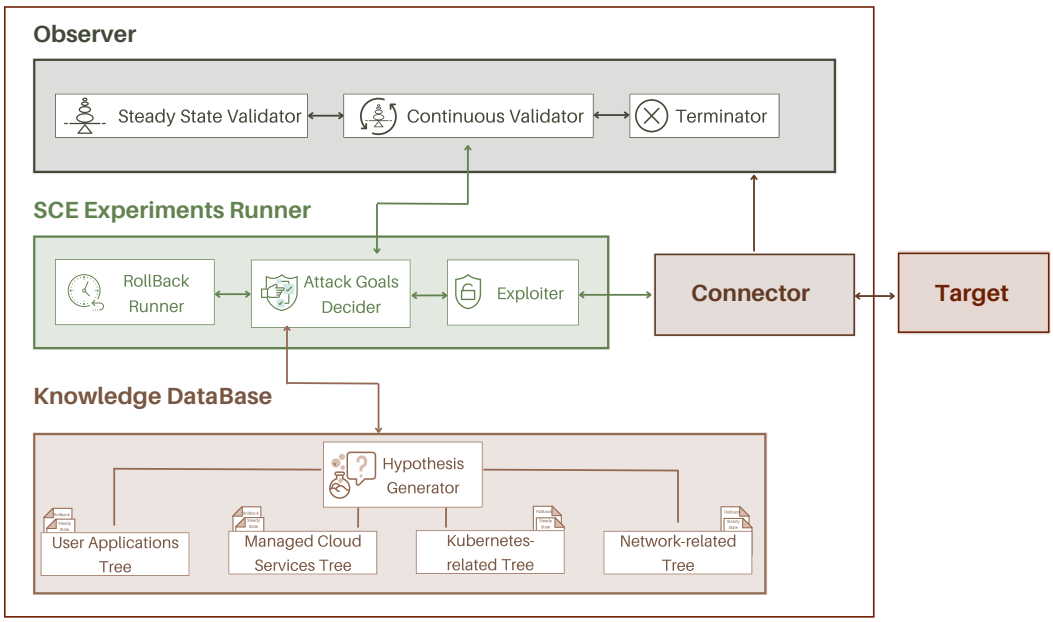

Figure 1: Proposed architecture of ChaosXploit

B. 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 for monitoring of 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.

1) Steady state Validator: The steady-state validator is in charge of verifying the steady-state hypothesis in the target that represents estable conditions. These 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.

2) 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 the target is not vulnerable and the following actions of the branch should not be executed.

3) 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 be able to inform the Rollback Runner so it can restore the target.

C. 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. It consists of three main elements: attack goal decider, exploiter, and rollback runner.

1) 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.

2) Exploiter: The exploiter executes the SCE experiment over a target to validate or refute 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.

3) 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.

D. Connector

The connector is responsible for searching for the most suitable extension to connect to the target on which the user

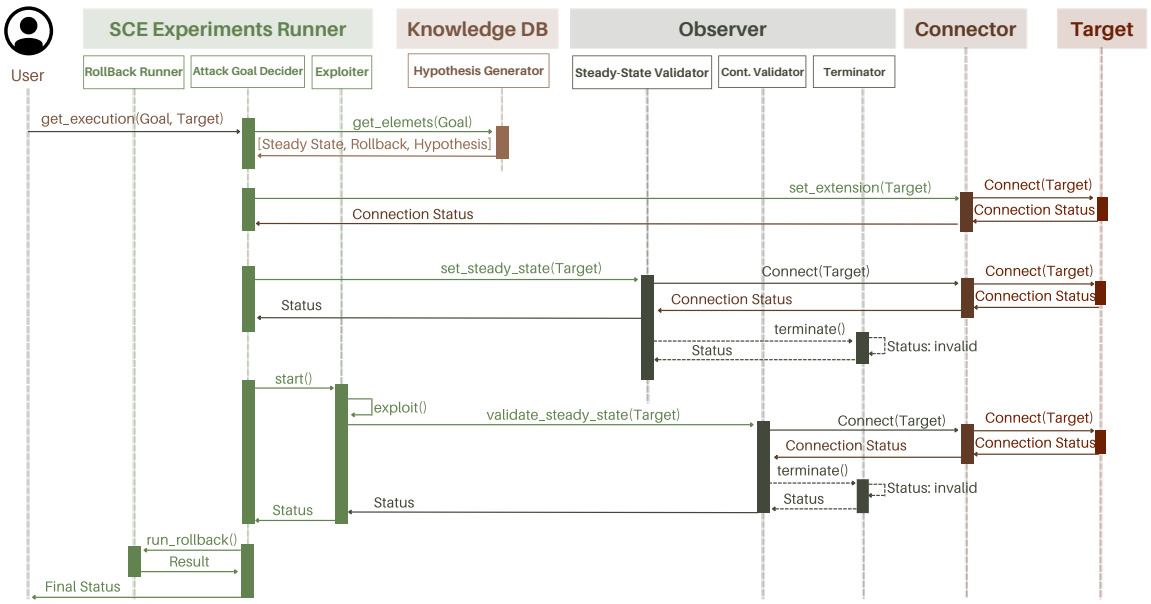

Figure 2: Flow diagram of the execution of a SCE experiment in ChaosXploit

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 interactions between the components of ChaosXploit are shown in Figure 2. First, the user of ChaosXploit requests 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 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, then the Attack Goal Decider does the following actions: i) sets the steady-state of the experiment in the Observer, ii) starts the execution of the steps defined in the first branch of the attack tree with the help of the Exploiter, and iii) keeps continuous communication with the Continuous Validator to monitor the execution of the exploitation in progress and in that way be aware of the attack goal was 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.

IV. Experiments

Multiple experiments have been conducted using the ChaosXploit proposal mentioned in Section III, which are also available in the repository of this project[3]. Based on the fact that AWS S3 buckets and Elasticsearch databases account for nearly 45% of the cloud misconfigured and compromised technologies [16], ChaosXploit focuses on evaluating the security of the AWS S3 service on this experiment. 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 is composed of the following subsections: Settings IV-A, where the hardware and software requirements to develop the experiment, are specified. Definition of the knowledge database IV-B, where the attack tree is presented together with the specification of the branch chosen for the experiment. SCE experiment IV-C in which the steady-state and the hypothesis of the experiment are defined, as well as the input parameters and the monitored variables. Finally, Results Analysis IV-D presents the results obtained.

A. 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 an open-source framework that

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

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.

B. Definition of the Knowledge Database

In Figure 3 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 inspect 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.
- **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 brute-forcing techniques to compromise admin 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 branch was included in the scope of this project, as the actions included in such branch were automatable completely. Other branches could also be implemented through a combination of manual and automatic actions.

C. 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 model 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:

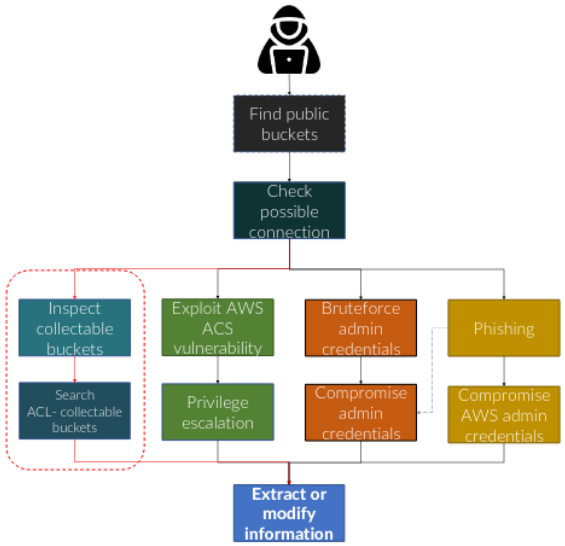

Figure 3: Attack Tree for the experimental scenario, highlighting the implemented path

- 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, we were able to clean up the buckets 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 I, different parameters were considered as input values for ChaosXploit. 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 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 I, ii) Buckets that have public ACLs, and can be accessed by anyone denoted by **ACL-Collectable** and iii) the **Permissions** obtained from the ACLs.

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

Table I: Monitored variables and input parameters for experiments.

D. 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[4], bucketkicker[5] or Sublist3r[6].

As seen in the upper left part of Figure 4, 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, e.g 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 act 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.

The lower left part of Figure 4 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 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 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 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. 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, 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 *AuthenticadedUsers* 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, 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.

[4]https://github.com/koenrh/s3enum
[5]https://github.com/craighays/bucketkicker
[6]https://github.com/aboul3la/Sublist3r

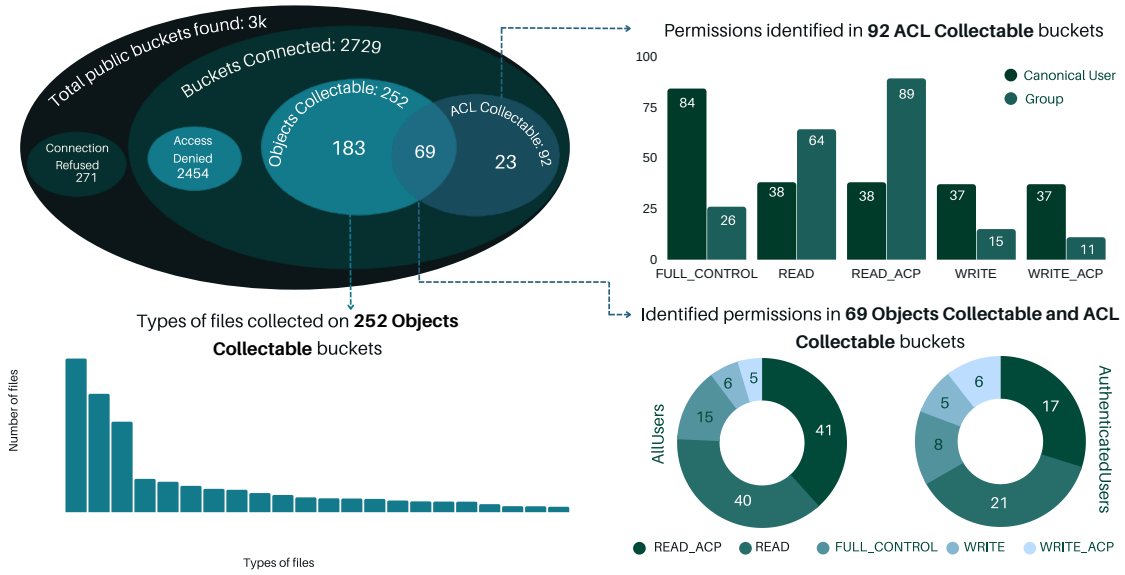

Figure 4: Results of the execution of each action included in the first branch of the attack tree

V. 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 [17] or threats addressed to a specific technology [18]. 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 cyber-attacks through continuous and rigorous experimentations on possible security holes and consequent mitigations.

In this paper, we proposed ChaosXploit, a SCE-powered framework that can 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 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 our approach can be successful, highlighting several unprotected buckets for a specific attack path. ChaosXploit was made publicly available for the cybersecurity community through the repository of the project[7].

Future work will explore the possibility of widening the ChaosXploit framework target architectures to include other use cases, systems, or providers. Besides, integrating a recommendation module to suggest countermeasures once a security flaw is discovered is worth investigating. Moreover, the performance of ChaosXploit should be further evaluated to prove its usefulness in performance-demanding scenarios.

Acknowledgment

This work has been supported by Universidad del Rosario (Bogotá) through the project “IV-TFA043 - Developing Cyber Intelligence Capacities for the Prevention of Crime”.

References

[1] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, *Site Reliability Engineering: How Google Runs Production Systems*, 1st ed. O’Reilly Media, Inc., 2016.

[2] A. Basiri, L. Hochstein, N. Jones, and H. Tucker, “Automating chaos experiments in production,” *CoRR*, vol. abs/1905.04648, 2019. [Online]. Available: http://arxiv.org/abs/1905.04648

[3] “Principles of chaos engineering,” https://principlesofchaos.org/, last time accessed: 2021-11-16.

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

[5] D. Díaz-López, M. Blanco Uribe, C. Santiago Cely, D. Tarquino Murgueitio, E. Garcia Garcia, P. Nespoli, and F. Gómez Mármol, “Developing secure iot services: A security-oriented review of iot platforms,” *Symmetry*, vol. 10, no. 12, 2018. [Online]. Available: https://www.mdpi.com/2073-8994/10/12/669

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

[6] D. Díaz-López, G. Dólera Tormo, F. Gómez Mármol, J. M. Alcaraz Calero, and G. Martínez Pérez, "Live digital, remember digital: State of the art and research challenges," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 109–120, 2014, 40th-year commemorative issue. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790613002905

[7] K. A. Torkura, M. I. Sukmana, F. Cheng, and C. Meinel, "CloudStrike: Chaos Engineering for Security and Resiliency in Cloud Infrastructure," *IEEE Access*, vol. 8, pp. 123 044–123 060, 2020.

[8] A. Basiri, N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds, and C. Rosenthal, "Chaos engineering," *IEEE Software*, vol. 33, no. 3, pp. 35–41, 2016.

[9] C. Camacho, P. C. Cañizares, L. Llana, and A. Núñez, "Chaos as a Software Product Line—A platform for improving open hybrid-cloud systems resiliency," *Software - Practice and Experience*, no. April 2021, pp. 1–34, 2022.

[10] J. Simonsson, L. Zhang, B. Morin, B. Baudry, and M. Monperrus, "Observability and chaos engineering on system calls for containerized applications in Docker," *Future Generation Computer Systems*, vol. 122, pp. 117–129, 2021. [Online]. Available: https://doi.org/10.1016/j.future.2021.04.001

[11] L. Zhang, B. Morin, P. Haller, B. Baudry, and M. Monperrus, "A Chaos Engineering System for Live Analysis and Falsification of Exception-Handling in the JVM," *IEEE Transactions on Software Engineering*, vol. 47, no. 11, pp. 2534–2548, 2021.

[12] A. Rinehart and K. Shortridge, "Security Chaos Engineering Gaining Confidence in Resilience and Safety at Speed and Scale," Tech. Rep., 2021.

[13] K. A. Torkura, M. I. Sukmana, F. Cheng, and C. 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.

[14] K. A. Torkura, M. Sukmana, F. Cheng, and C. Meinel, "Continuous auditing and threat detection in multi-cloud infrastructure," *Computers and Security*, vol. 102, p. 102124, 2021. [Online]. Available: https://doi.org/10.1016/j.cose.2020.102124

[15] S. Sharieh and A. Ferworn, "Securing apis and chaos engineering," in *2021 IEEE Conference on Communications and Network Security (CNS)*, 2021, pp. 290–294.

[16] Rapid7, "2021 cloud misconfiguration report," 2021.

[17] I. Martínez Martínez, A. Florián Quitián, D. Díaz-López, P. Nespoli, and F. Gómez Mármol, "Malseirs: Forecasting malware spread based on compartmental models in epidemiology," *Complexity*, vol. 2021, 2021.

[18] P. Nespoli, D. Díaz-López, and F. Gómez Mármol, "Cyberprotection in iot environments: A dynamic rule-based solution to defend smart devices," *Journal of Information Security and Applications*, vol. 60, p. 102878, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214212621001058