Security Chaos Engineering for Cloud Services

Conference Paper · September 2019

DOI: 10.1109/NCA.2019.8935046

4 authors:

Kennedy Torkura
Hasso Plattner Institute
37 PUBLICATIONS **134** CITATIONS
SEE PROFILE

Muhammad Ihsan Haikal Sukmana
Hasso Plattner Institute
28 PUBLICATIONS **105** CITATIONS
SEE PROFILE

Feng Cheng
Peking University
104 PUBLICATIONS **1,435** CITATIONS
SEE PROFILE

Christoph Meinel
Hasso Plattner Institute (Potsdam Germany)
1,322 PUBLICATIONS **10,793** CITATIONS
SEE PROFILE

Security Chaos Engineering for Cloud Services
(Work In Progress)

Kennedy A. Torkura, Muhammad I.H. Sukmana, Feng Cheng and Christoph Meinel
Hasso-Plattner-Institute for Digital Engineering,
University of Potsdam,
Potsdam, Germany
Email: firstname.lastname@hpi.de

***Abstract*—The majority of security breaches in cloud infrastructure in recent years are caused by human errors and misconfigured resources. Novel security models are imperative to overcome these issues. Such models must be *customer-centric*, *continuous*, not focused on traditional security paradigms like intrusion detection and adopt *proactive* techniques. Thus, this paper proposes *CloudStrike*, a cloud security system that implements the principles of *Chaos Engineering* to enable the aforementioned properties. Chaos Engineering is an emerging discipline employed to prevent non-security failures in cloud infrastructure via *Fault Injection Testing* techniques. CloudStrike employs similar techniques with a focus on injecting failures that impact security i.e. integrity, confidentiality and availability. Essentially, CloudStrike leverages the relationship between dependability and security models. Preliminary experiments provide insightful and prospective results.**

***Index Terms*—Cloud-Security, Security Chaos Engineering, Resilient Architectures, Security Risk Assessment**

I. Introduction

Chaos Engineering [1] is a discipline that emphasizes on intentional injection of *faults* into software systems to minimize downtime while increasing resiliency. The main motivation for this approach is to overcome uncertainties prevalent in distributed systems e.g. cloud infrastructure. Hence, companies that employ chaos engineering principles e.g. Netflix operate fault-tolerant environments on public clouds. However, similar achievements are yet to be attained for with cloud security, rather security breaches are increasing. Interestingly, a large proportion of these breaches are caused by human errors, e.g. misconfigured Access Control Policies (ACP) and provisioning of over-privileged users. The Cloud Security Alliance (CSA) asserts that the two most severe cloud security issues in 2019 are *data breaches* and *misconfiguration and inadequate change control* [2]. This fact is collaborated in the Ponemon Institute's Data Breach Report for 2019, which alleged that 49 % of breaches are caused by system glitches and human errors [3]. Since most of these breaches result from human errors, it is imperative to evolve security models that potentially consolidate the *human factor* in cloud security chain. Therefore, automated, customer-centric, continuous and proactive security models are required to overcome these contemporary security challenges.

This *work-in-progress* paper presents *CloudStrike*, a cloud security system that employs chaos engineering principles to address the aforementioned security challenges. Preliminary versions of CloudStrike (then called *BrokerMonkey*) were used in our previous work to test the efficiency of CSBAuditor [4] and SlingShot [5]. These systems were designed for threat detection and incident response in cloud systems, and CloudStrike generated dynamic workloads (malicious attacks) for the experimental evaluation. In these scenarios, CloudStrike was used in very specific ways, thus we aim at designing it to be open, flexible and suitable for orchestrating dynamic, security workloads against heterogeneous cloud systems. Within this context, the following research questions are relevant:

- How can continuous security techniques be combined with the principles of chaos engineering to improve cloud security?
- What existing security testing principles can be leveraged with chaos engineering?
- How can the existing models that combine dependability and security be leveraged for security chaos engineering?

The rest of this paper is structured as follows, in the next section, we discuss cloud security challenges. Section III presents the design and implementation of CloudStrike while preliminary results of our evaluation is in Section IV. The paper concludes in Section V with a brief discussion on future work.

II. Cloud Security Challenges

Attacks against cloud infrastructure have increased in recent years. The most targeted cloud services are cloud storage (e.g. AWS S3) and Identity and Access Management (IAM). These attacks result from misconfiguration errors and other forms of human error [2]. Several industry reports e.g. CSA [2], Ponemon Institute[3] and OWASP [6] have highlighted the impact of misconfiguration errors on cloud infrastructure. Examples of these errors include misconfigured cloud storage bucket policies, ACLs and role policies. Thus, overcoming these challenges is an open research challenge, especially as CCs are responsible for these aspects as defined in the Shared Security Responsibility Model (SSRM). Thus, tools to overcome these issues should be *customer-centric* i.e. designed to overcome human errors by specifically considering the CCs and automating related human tasks.

Similarly, failures (e.g. out of bounds array accesses) often affect availability of services including nodes and networks. To prevent these failures, *chaos engineering* principles adopt Fault

978-1-7281-2522-0/19/$31.00 ©2019 IEEE

Injection Testing (FIT) techniques to intentionally introduce failures [7] in production systems. This affords better insights into failure scenarios and aids the design and implementation of fault-tolerant systems. However, Al-Kuwaiti [8] suggests that both fault-tolerance and security are important properties for *dependability*. This implies that current chaos engineering system (e.g. Netflix Chaos Monkey [1]) do not satisfy dependability requirements due to lack of security properties. This gap can be addressed by extending chaos engineering principles to security i.e. *security chaos engineering*. Techniques similar to FIT can be leveraged to inject failures that impact on security: confidentiality, integrity and availability, and potentially address the challenge of security incidents resulting from human errors and misconfigured resources.

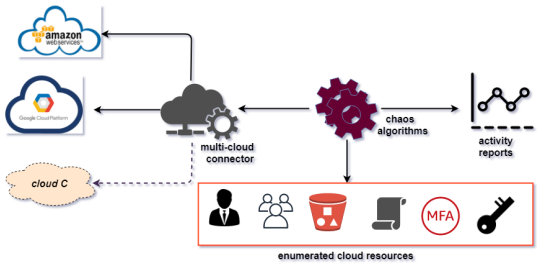

Figure 1: High Level Overview of CloudStrike

III. CloudStrike: Security Chaos Engineering for Cloud Services

CloudStrike (Figure 1), employs the chaos engineering principles proposed by Basiri et.al [1] as explained below:

A. Build A Hypothesis Around a Steady-state Behavior

Central to every chaos engineering experiment is the determination of a hypothesis about *normalcy* and *abnormality*, with corresponding measurable attributes. Thus, we exploited the concept of *expected state* [4] - the secure state of a cloud resource at time t_o. Essentially, the expected state is known by the resource orchestration system. For example, an ACP may specify *read* access for a user, *Alice* at time t_o. This is registered in the orchestration system and a measurable attribute is defined e.g. a HTTP 401 error (*unauthorized*) is produced if Alice sends a request to a resource (e.g. bucket) after her privileges are removed.

B. Vary Real World Events

In order to simulate real world events, a variation of possible attacks is implemented. CloudStrike orchestrates random actions against target cloud systems e.g. deletion, creation, and modification, using the respective cloud APIs. Three chaos modes are supported: *LOW*, *MEDIUM* and *HIGH*, which correspond the magnitudes of 30 %, 60 % and 90 % respectively. Table I is an example of *AttackPoints* used, each AttackPoint defines a specific action to be conducted, a combination of two or more AttackPoints forms an *attack scenario*. Algorithm 1

[1]https://github.com/Netflix/chaosmonkey

Algorithm 1 Malicious-User-Bucket-Attack-Scenario

1: **procedure** BUCKETATTACK
2: *createNewUser()* ▷ create a new user e.g. Bob
3: *getCloudBuckets()* ▷ get a list of all the buckets in the cloud
4: $selectRandomBucket \leftarrow getCloudBuckets()$ ▷ select a random bucket from the set of buckets in the cloud
5: *createBucketPolicy()*
6: $assignUserAccessPolicy \leftarrow selectRandomBucket$ ▷ give user e.g. Bob read access to the existing bucket
7: **end procedure**

Table I: Examples of CloudStrike's AttackPoints

ID	Cloud Resource	Chaos Action	Description
AP1	User	create	create random user
AP2	User	delete	delete existing user
AP3	User	modify	change user configuration e.g. privileges, role or group
AP4	Policy	create	create new policies with random ACLs and attach to cloud resource(s)
AP5	Policy	modify	modify existing policy e.g. change ACL to deny original owner access to the resource
AP6	Policy	delete	detach policy from a resource, delete the policy
AP7	Role	create	creae a new role
AP8	Bucket	make public	alter private configuration to public
AP9	Bucket	disable logging	stop logging API calls against bucket
AP10	Bucket	make unavailable	simulate bucket unavailability e.g. by changing bucket ACL from ALLOW to DENY

combines $AP1$ and $AP4$ to create a scenario where an attacker creates a random user in a cloud account, creates a privileged policy for accessing a cloud bucket and attaches the policy to the malicious account.

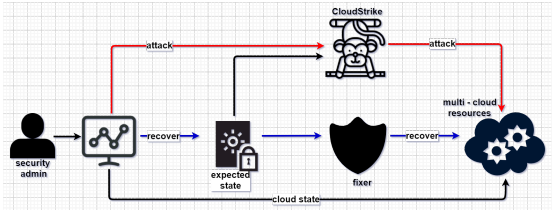

Figure 2: CloudStrike Overview Showing Safety Measures

C. Run Experiments in Production

Chaos engineering experiments are initially run in development environments to ascertain the outcomes are as designed to ensure safety. However, the experiments are to be deployed against production environments to derive the value of the ideology. While this might sound scary initially, this is a foundational approach for building antifragile systems [9]. However, as a best practice, safety measures are required e.g. for recovering systems to *secure states*. We achieve this by employing the concept of *expected states* and *cloud state*[4], illustrated in Figure 2. These expected states are persisted, and can be easily used to recover cloud environments to its secure states. We deployed CloudStrike against resources deployed

```
{
   "buckets":{
      "created":[
         "bucket-23627307",
         "bucket-7635123e"
      ],
      "loggingDisabled":[
         "company-baumbach-log-c1d5eb38",
         "company-schumm-log-f95d397c"
      ],
      "madePublic":[
         "company-bergnaum-data-14eeac91"
      ],
      "accessDenied":[
         "company-kiehn-log-18f0e65f",
         "company-doyle-data-2fb7a1be"
      ]
   },
   "serviceAccounts":{
      "created":[
         "user_6217f6e1-7cd5",
         "user_47a1a220-b42c"
      ],
      "deleted":[
         "company-checker-log",
         "company-howe-log"
      ],
      "grantedPermission":[
         "company_turcotte_log",
         "company_osinski_file"
      ]
   }
}
```


Figure 3: Summary Report for an Attack Against AWS

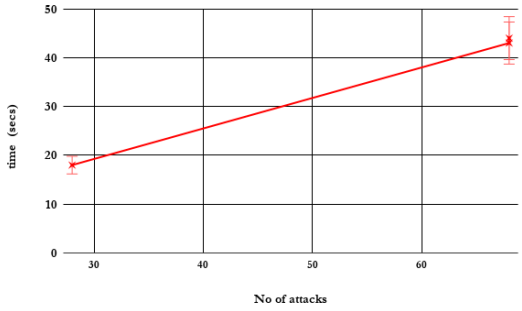

Figure 4: Performance of CloudStrike against for 50 Users

on Amazon Web Services (AWS) and Google Cloud Platform (GCP), Figure 3 is a report of an experiment.

D. Automate Experiments to Run Continuously

A clear distinction between traditional security testing and chaos engineering is the use of automation. Security automation enables continuous oversight, which is necessary in the cloud due to constant changes e.g. change of ACPs and provisioning of new API keys. These changes could be initiated for either malicious or benign reasons hence the need for proactively measures e.g. security chaos engineering approaches to experiment and study malicious scenarios, thereby gaining insights into efficient ways for designing and implementing secure cloud systems.

IV. Preliminary Performance Evaluation

We conducted preliminary experiments to evaluate CloudStrike using assets on AWS and GCP. CloudStrike orchestrates several kinds of attacks e.g. modification of bucket ACP (render unavailable), API calls are responded to with error code 404. Similarly, CloudStrike misconfigures buckets by changing Access Control List (ACL) from *PRIVATE* to *PUBLIC*. CloudStrike's *attack rate* [10] i.e. the number of attacks launched per second is measured. 50 user accounts are provisioned on both cloud platforms i.e. 25 per cloud. Each user account is properly configured using privilege separation concepts. CloudStrike then enumerates the cloud accounts, and acquires detailed information on composition and deployment. Appropriate attacks are then composed, based on our algorithms (e.g Algorithmn 1). The attacks are launched against the cloud accounts, while recording the time-taken from start to completion. The procedure is repeated in increasing magnitudes of *LOW*, *MEDIUM* and *HIGH* (corresponding to magnitudes of 30 %, 60 % and 90 % respectively). The algorithms implement random actions e.g. creation, deletion and modification of resources. Figure 4 illustrates the performance of a *LOW* mode. We observe that the time taken linearly increases proportional to number of target workload. The time taken has no significant overhead to the system hence it can be scaled to attack enterprise clouds comprising hundreds of assets.

V. Conclusion and Future Work

We have presented *CloudStrike*, a security chaos engineering system designed for multi-cloud security. CloudStrike leverages chaos engineering principles with a focus on security by injecting faults that impact confidentiality, integrity and availability of cloud resources. A limitation is that, cloud buckets contents cannot be deleted or modified since they might be lost completely. For future work, we aim at extending CloudStrike e.g. using JSON files for configuring the chaos modes and increasing observability with user interfaces. It will also be interesting to research more on existing models that combine dependability and security.

References

[1] A. Basiri, N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds, and C. Rosenthal, "Chaos engineering." *IEEE Software*, 2016.

[2] "Top threats to cloud computing the egregious 11," *Cloud Security Alliance*, 2019.

[3] "2019 cost of a data breach report," Online, 2019, [Accessed: 08 September 2019].

[4] K. A. Torkura, M. I. Sukmana, T. Strauss, H. Graupner, F. Cheng, and C. Meinel, "Csbauditor: Proactive security risk analysis for cloud storage broker systems," in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2018, pp. 1–10.

[5] ——, "Slingshot: Automated threat detection and incident response in multi cloud storage systems (to appear)," in *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2019.

[6] OWASP, "Application security risks-2017. open web application security project (owasp)," 2017.

[7] K. Andrus, N. Gopalani, and B. Schmaus, "Fit: Failure injection testing," *Netflix Tech Blog*, vol. 23, 2014.

[8] M. Al-Kuwaiti, N. Kyriakopoulos, and S. Hussein, "A comparative analysis of network dependability, fault-tolerance, reliability, security, and survivability," *IEEE Communications Surveys & Tutorials*, vol. 11, no. 2, pp. 106–124, 2009.

[9] S. Newman, *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.", 2015.

[10] J. Mirkovic and P. Reiher, "A taxonomy of ddos attack and ddos defense mechanisms," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.